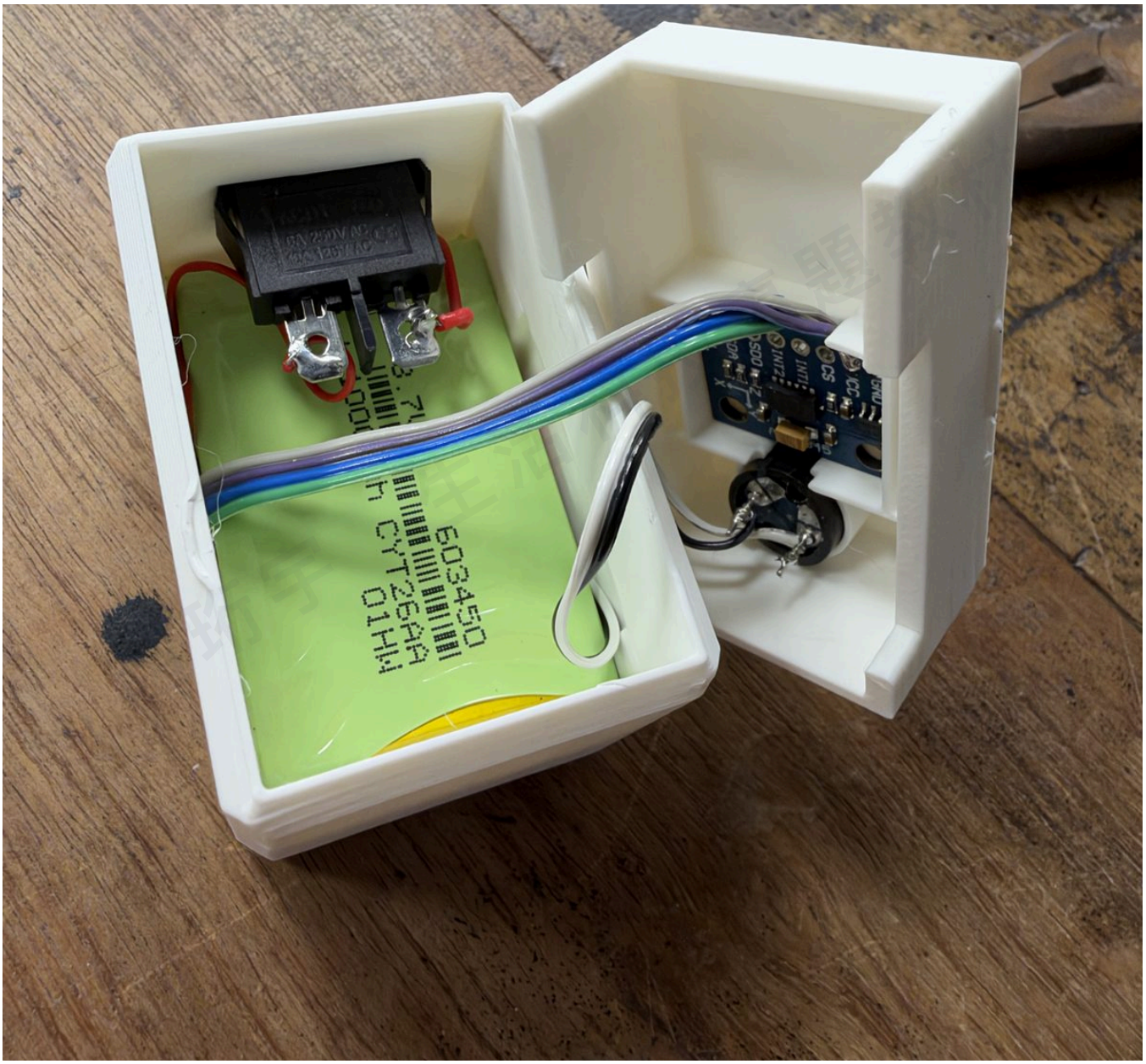
自製計步器
從硬體、演算法到模擬器

TTGO T-Display × ADXL345 × 蜂鳴器｜中文版韌體與瀏覽器模擬器｜教學講義



生活科技　專題製作教材

趙珩宇　2026 年 10 月

目錄

這份講義怎麼讀：每一節先看「它做了什麼」，再講「為什麼」，最後是「所以我們要注意什麼」。

1　先看成果：這台裝置會做什麼

2　硬體解剖：每個零件在做什麼

3　接線與原因：為什麼接這幾支腳

4　從加速度到步數：演算法怎麼想

5　模組比較與選型：換一顆行不行

6　軟體架構與模擬器：先在電腦上跑

7　失敗路徑：我們踩過的坑

8　兩個逆向案例：Pico 成功、ESP32 停下

9　延伸專案（七個，程式在模擬器網站）

10　遊戲化方向：卡帶架構、三種待機主題、A / B 兩條路線

11　專案歷程：時間軸、關鍵決策、AI 的設計方法、未完成事項

12　逐步實作步驟（14 步，每步有驗收）

13　第二輪改良：USB 進時鐘、自動對時、狀態列、Wi-Fi 開關、自動關螢幕、音樂、螢幕翻轉（做了又拿掉）

14　計步演算法第二版：走路測試、諧波的誤判、自相關

15　效能與省電：量測工具、五項改善、FIFO、light sleep、記憶體分區

16　藍牙與網頁工具：BLE 與手機連線、瀏覽器燒錄、OTA

附錄　腳位表、API、指令、材料包、官方資料、除錯流程、審稿紀錄、圖片出處、**I 名詞解釋**

這份講義的來源

原作是學生專題「GOTT Pedometer」（英文介面）。學生電腦故障、原始碼遺失，我們先把裝置的快閃記憶體做了唯讀備份與分析， 再以原作的功能為規格重新設計中文版，並做出一個在瀏覽器裡就能跑的模擬器。第 8 節會說明「為什麼原始碼抓不回來」。

建國科技大學 生活科技專題教材

需要準備

硬體：LILYGO TTGO T-Display V1.1、ADXL345（GY-291）模組、無源蜂鳴器、3.7 V 鋰電池（1000 mAh，含開關）、杜邦線、USB-A 轉 USB-C 線。

軟體：任何能開網頁的電腦（模擬器）；要燒錄時需 Arduino ESP32 核心 3.3.x（本專案用 arduino-cli）。

趙珩宇　生活科技專題教材

1　先看成果：這台裝置會做什麼

開機或插上 USB 後是待機時鐘，任一鍵進入五項的中文主選單：計步、音樂、七日紀錄、遊戲（第 10 節）、設定。主選單一次只顯示一項：左邊是會動的小插圖，右邊的轉盤提示目前轉到第幾項。兩顆板載按鍵就是全部的操作。橫向拿時兩顆鍵在同一側、上下排列：上鍵「上一項 / 加」、下鍵「下一項 / 減」、兩鍵同按短放「進入 / 確認」、兩鍵同按長按「回上一層」（在主選單長按則直接回時鐘）。這一節是第一版的功能；v1.1～v1.8 加上的功能（音樂、藍牙、螢幕翻轉、自動關螢幕、網頁燒錄等）在第 13～16 節。



主選單（v1.8）：一次顯示一項，左邊是動畫插圖與今天的步數，右邊轉盤上是相鄰的項目；上下鍵轉動、同按進入。



計步：步數、每日目標進度條、估算卡路里與距離。找不到加速度計時會直接寫在畫面上。



音樂播放中：曲名、出處、進度條與音量。停止時上下選歌、同按播放；播放中上鍵大聲、下鍵小聲，同按停止（13.5 節）。

功能	做什麼	原作有嗎 / 新版補了什麼
計步	以 ADXL345 的加速度判斷步伐，算步數、距離（步幅）、卡路里（體重）	原作有。新版把步幅、體重、目標、演算法參數都做成可調
音樂	用蜂鳴器播放 9 首公眾領域的旋律；網頁也能點歌	原作是番茄鐘。v1.7 依使用回饋移除番茄鐘，改成音樂（13.5、13.6 節）
水平儀（已移除）	用重力方向算俯仰（pitch）與橫滾（roll）角	原作有。v1.9.4 依老師回饋拿掉：每個人感測器的安裝方向不同，讀數不一致（13.6 節）
七日紀錄	每天的步數；另有每分鐘步數紀錄（30 天），儀表板畫成每小時長條	原作有欄位但沒有時鐘、不會換日。新版對時後真正能換日
無線網路與網頁	裝置自己開熱點；螢幕顯示 **Wi-Fi QR Code**，手機相機掃一下就加入，連上後**強制門戶**讓儀表板自動跳出（不必打網址）	原作有，英文、無密碼、要手動輸入 SSID 與網址。新版中文、掃碼即連、自動跳頁、會改資料的操作才要密碼、顯示電量

待機時鐘	開機、插上 USB 與閒置 30 秒都進入「上面大字時刻、下面一條會動的關卡」畫面（參考 Game & Watch 時鐘的結構）：角色在地面上跑，走路時場景加速捲動、靜止時慢慢散步；右邊掉障礙物、自動跳過；每整點撞問號方塊噴三枚金幣＋金幣音，數字變金色；每滿 100 步噴一枚金幣。五種主題（小跑者 / 小恐龍 / 貓咪 / 台鐵 / 新幹線）在設定選單切換，每種主題的背景、障礙物與整點事件都不同。右側直排電池、Wi-Fi、藍牙圖示，充電時有閃電。背光降到 30%，計步照跑；任一鍵回選單	原作無。時間來自插電腦時自動對時、藍牙或手機開網頁（13.1 節）
設定選單	無線網路（QR）、Wi-Fi 開關（自動 / 完全關閉）、藍牙、個人資料（性別 / 年齡 / 身高 / 體重 / 步幅）、蜂鳴器音量（關 / 小 / 中 / 大）、螢幕亮度（30 / 50 / 100%，預設 50）、自動關螢幕、待機主題、走路測試、操作說明、關於、關機	原作無。卡路里改依個人資料計算，見 4.4
關機	設定→「關機」→ 同按確認 → 存檔、時間寫入 RTC → 深度睡眠；上鍵開機，時間接回	原作是剪電池紅線串實體開關。新版不裝開關、不剪電池線（第 2.2 節）

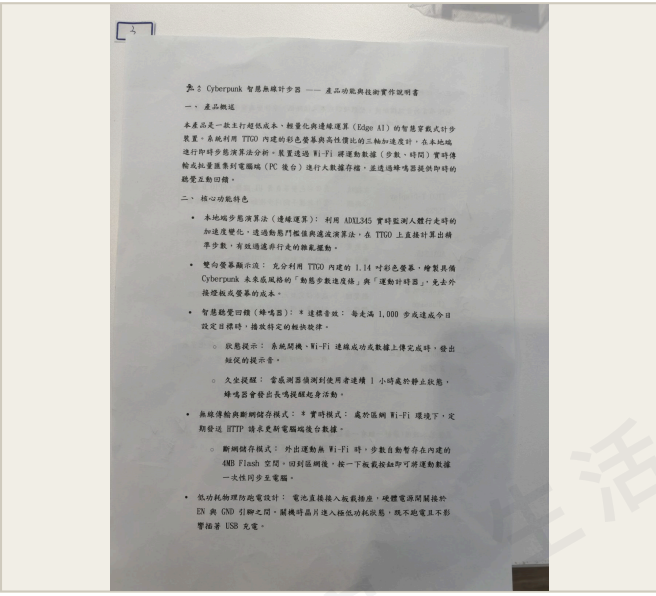
1.1 學生的說明書寫了什麼、韌體實際做了什麼

原作附有一份「Cyberpunk 智慧無線計步器——產品功能與技術實作說明書」。把它和從韌體裡取回的字串、分區、NVS 內容對照，會發現文件與實作已經「漂移」——這在真實專案裡非常常見，也是這份講義以**韌體**而非說明書為規格的原因。

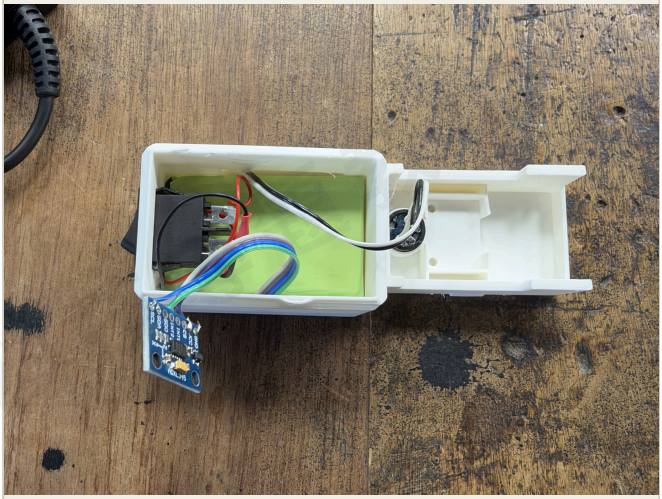
說明書寫的	韌體裡找到的	怎麼解讀
動態門檻＋濾波的本地步態演算法	有計步與自動存檔；參數細節在機器碼裡看不到	一致，但無法驗證「動態門檻」是否真的實作
Cyberpunk 風格步數進度條與運動計時器	有 `step_goal` 進度與番茄鐘（WORK / BREAK）；畫面文字全英文	「運動計時器」做成了番茄鐘
每 1,000 步或達標播放旋律；開機 / 連線 / 上傳提示音	有 `toneTask`；沒有任何「上傳」相關字串	有響鈴，但沒有上傳可提示
久坐 1 小時長鳴提醒	字串中完全沒有相關訊息	很可能沒做，或做了沒留訊息

逢甲大學 生活科技 實習教材

實時模式：定期 HTTP 請求上傳到電腦後台	**方向相反**：裝置自己開熱點當伺服器，瀏覽器來連；沒有任何外部網址	架構在開發中被改掉了，文件沒更新
斷網時暫存 4 MB Flash，按板載按鈕一次同步到電腦	步數存在 NVS，有七日欄位 `h_step_` / `h_pomo_`（但沒有時鐘所以沒資料）；沒有同步功能	「存」有做，「同步」沒做
電源開關接在 EN 與 GND 之間	實物照片：開關**串在電池正極線**上	兩種接法各有取捨，見 2.1
（沒寫）	水平儀、七日紀錄畫面、網頁儀表板	做了沒寫進文件



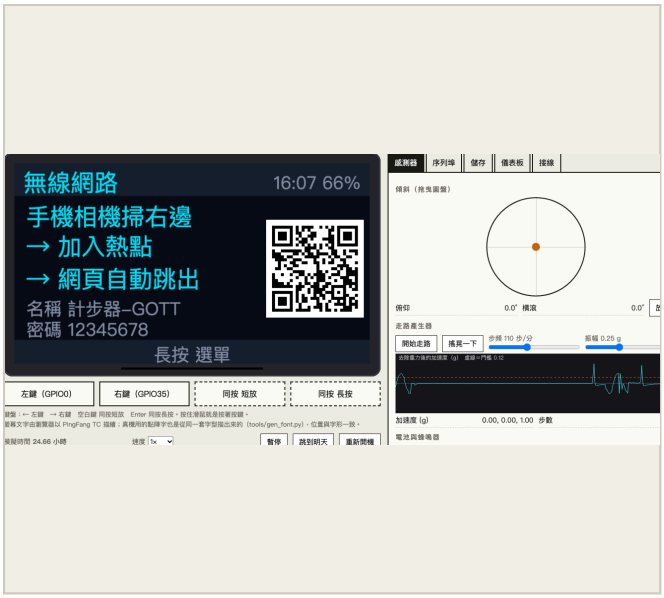
原作說明書第一頁。寫作時間早於最終韌體，所以有多處與實作不符。



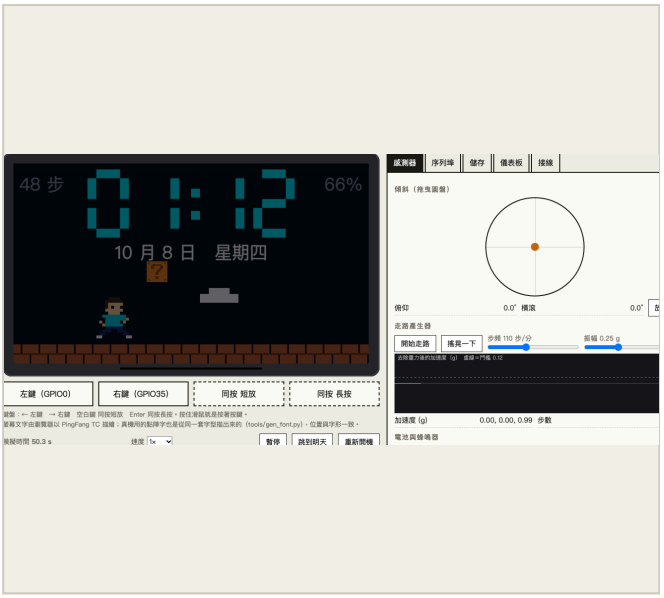
原作拆開後的全部零件：T-Display、ADXL345（GY-291）、無源蜂鳴器、1000 mAh 鋰電池、船型開關、3D 列印外殼。

教學點

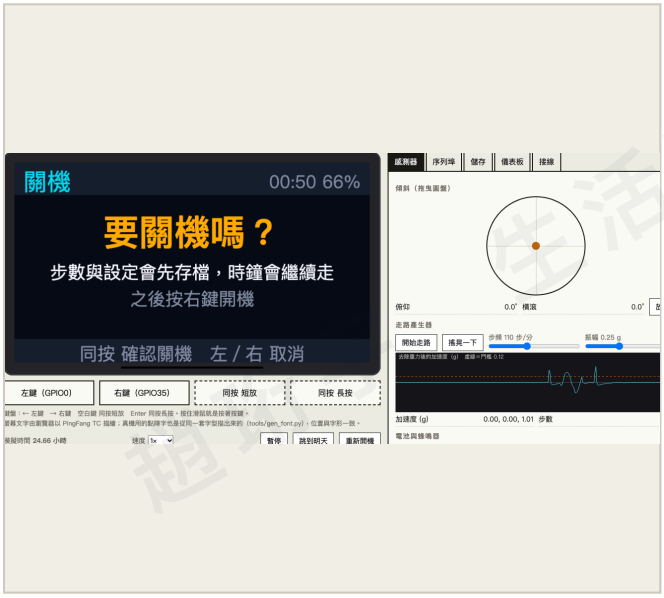
說明書是「打算做什麼」，韌體是「真的做了什麼」。交件時要回頭用實作校正文件；拿到別人的專案時，先信實物與程式，再信文件。



無線網路畫面：右側是標準 Wi-Fi 設定 QR（`WIFI:T:WPA;S:…;P:…;;`），iPhone / Android 相機掃到就問要不要加入。連上後手機偵測到「沒有網際網路」，裝置的 DNS 把所有網域答成自己、探測網址一律轉向儀表板，手機就自動彈出登入頁——跟飯店 Wi-Fi 一樣的機制。



待機時鐘：上面 7 段大字（只用矩形畫），下面是會捲動的關卡——問號方塊、水管、雲、磚地與原創的小跑者（12×16 像素圖放大 2 倍）。結構參考 Game & Watch 時鐘，角色與場景皆自繪。

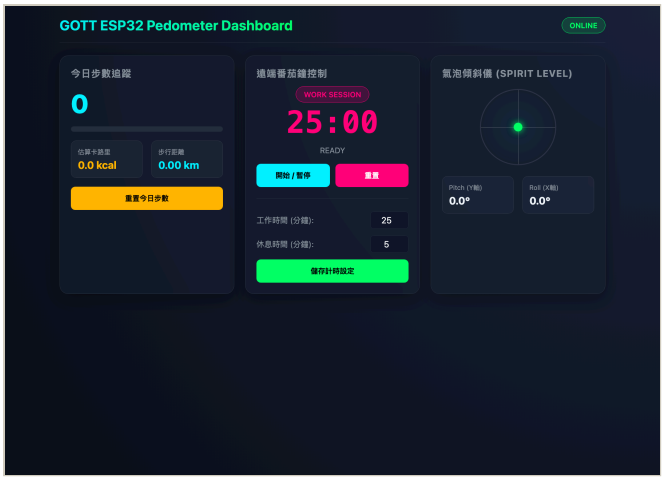


主選單「關機」的確認畫面：同按確認後存檔、時間寫進 RTC、深度睡眠；上鍵開機（舊版截圖寫「右鍵」）。



（第 10 節）擲筊卡帶。

超級生活科技專題教材



原作的網頁儀表板（從韌體裡取回的原始 HTML）：英文介面、深色卡片。



新版儀表板在模擬器裡執行，全中文。截圖為 v1.6 以前的版本；現在是狀態、音樂、七日紀錄、設定四個分頁（番茄鐘 v1.7 換成音樂、水平儀 v1.9.4 拿掉）。

趙珩宇 生活科技專題教材

2　硬體解剖：每個零件在做什麼

零件	角色	關鍵規格（為什麼重要）
TTGO T-Display 主控＋螢幕	ESP32 雙核微控制器，板上就有 1.14 吋彩色螢幕、兩顆按鍵、USB 轉序列晶片、鋰電池充電與電池座	3.3 V 邏輯；4 MB 快閃記憶體（預設程式分區只有 1.2 MB，第 7 節會踩到；現在改用 min_spiffs 分區，見 15.5）；Wi-Fi 開啟時耗電 80～240 mA
ADXL345 三軸加速度計	量三個方向的加速度，靜止時量到的是重力（1 g）。走路的上下震動、傾斜時重力方向的改變，都靠它	±2～16 g 可選；每格 3.9 mg；I2C 位址 0x53；GY-291 模組板上有穩壓器與上拉電阻
無源蜂鳴器	播放音樂、按鍵回饋、達成目標與整點時發聲	「無源」表示內部沒有振盪器，要由 GPIO 送方波（`tone()`）才會響，所以能選音高、放旋律
鋰電池 1000 mAh ＋電源開關	離開 USB 也能用。開關串在正極線上	3.7 V 標稱、4.2 V 滿電、3.3 V 以下應停用；接 T-Display 的 JST 電池座，板上 TP4054 會在插 USB 時充電

鋰電池安全守則（實作前先讀）

① **不要同時剪紅黑兩條線**：斜口鉗刀口會讓正負極瞬間短路。一次只處理一條：剝皮→套熱縮管→焊→烘縮→確認絕緣，再處理另一條。② 開關的裸露接腳一定要用熱縮管包好，塞進外殼時才不會刮破電池的鋁塑膜（照片裡原作的開關焊點就是裸露的，是要改進的地方）。③ **電壓不要低於 3.0 V**；本韌體把 3.3 V 顯示為 0%，看到 0% 就關機充電。④ 電池鼓起、發熱、外膜破損一律停用。⑤ 充電時人要在場。

2.1 電源：電池直接插電池座，不裝實體開關

是什麼　T-Display 的 JST 座後面接著充電晶片（TP4054）與電源切換電路；電池紅黑線直接插上去。新版**沒有實體開關**：關機是設定選單裡的一個項目（深度睡眠），按上鍵開機。

為什麼　原作是剪斷電池紅線串一顆船型開關。它能動，但「剪線那一刀」與裸露的焊點是鋰電池最大的風險；而且電池成了專用品，壞了要重焊。我們比較了四種做法（下表），最後選擇像 Flipper Zero 一樣用軟體關機：不碰電池線、不多一個會壞的零件、關機時仍能充電、時鐘由 RTC 繼續走。

所以呢　代價是關機時仍有約 0.1～2 mA 的耗電（最大宗是 GY-291 模組上的紅色 LED）；程式當掉時的最後手段是拔電池。這兩點對學生裝置都可以接受，也是很好的討論題。

關機方式	關機時	優點	代價	採用
剪電池紅線串開關（原作）	電池與板子完全斷開，0 μA	真正斷電；接線直覺	剪線短路風險；關機時插 USB 不會充電；電池變專用品	否
JST 轉接線串開關	同上	不剪原廠電池線；電池可整顆更換	多一組接頭；關機時不充電	備案
開關接 EN—GND（原作說明書）	只有 ESP32 停（約 1 μA），穩壓器、模組 LED、ADXL345 仍耗約 2～3 mA	關機能充電；不碰電池線	EN 沒拉到排針，要焊在 RST 側鍵的焊點；燒錄時開關須打開	否
軟體關機：選單→深度睡眠	ESP32 約 10 μA；螢幕 Sleep In、ADXL345 待機、分壓關閉；板上穩壓器約 55 μA、模組 LED 1～2 mA 仍在	不裝開關、不碰電池線；關機能充電；**RTC 繼續計時**，開機接回	程式當掉只能拔電池；開機約 1～2 秒	**採用**

想更省電：拆掉 GY-291 上 LED 的限流電阻，關機電流降到約 0.1 mA，1000 mAh 可放半年。這是延伸專案 D 的實驗。

2.2 有源蜂鳴器與無源蜂鳴器

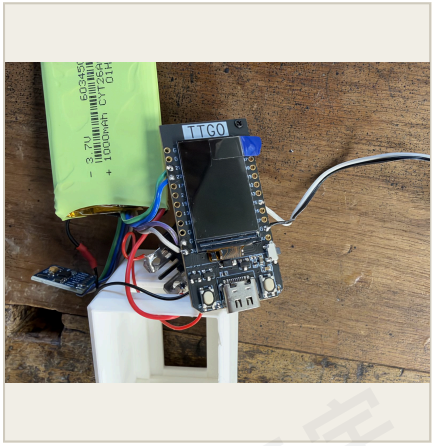
	有源（Active）	無源（Passive）←本專案
內部	有振盪器，給直流就響	只有壓電片或線圈，要外部送方波
程式	`digitalWrite(HIGH)`	`tone(pin, 頻率, 時間)`

能做的事	單一音高的嗶聲	任意音高、旋律、音效
怎麼分辨	外觀很像。背面被黑膠封住、有「＋」記號的多半是有源；看得到綠色電路板或線圈的多半是無源。最保險：接 3.3 V 直流，會響的是有源。	
接反會怎樣	有源有極性，接反不響	無源沒有嚴格極性，接反照樣響；習慣上「＋」接 GPIO

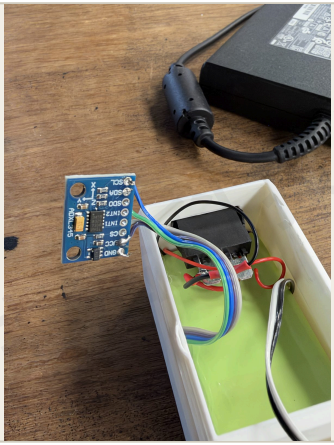
常見誤判

學生把有源蜂鳴器接上 `tone()`，結果只有「沙沙聲」或固定音高——因為內部振盪器會蓋過外部方波。反過來把無源蜂鳴器接直流，則完全不響。

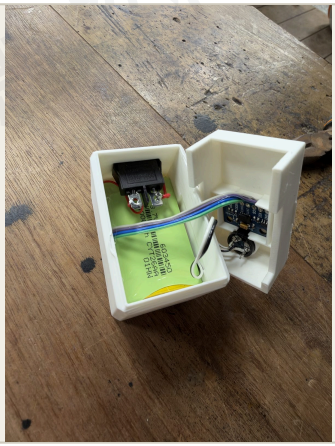
2.3 實物長這樣



T-Display 背面：排線焊在 21、22、G、3V；蜂鳴器白線焊 25、黑線焊 G；電池紅黑線進 JST 座。



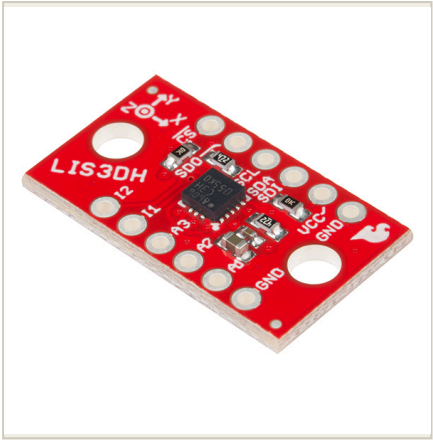
GY-291 ADXL345：只焊 SCL（藍）、SDA（綠）、VCC、GND 四支腳；CS、SDO、INT1、INT2 不接。



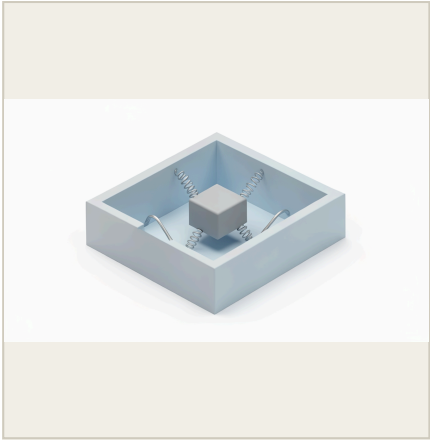
（原作）外殼下半部放船型開關與電池，上蓋放 ADXL345 與黑色圓殼的無源蜂鳴器，排線穿過側邊缺口。新版不裝開關。



替代型號 MPU6050（GY-521）：加速度＋陀螺儀，位址 0x68。



替代型號 LIS3DH（SparkFun 版）：超省電，穿戴裝置常用。



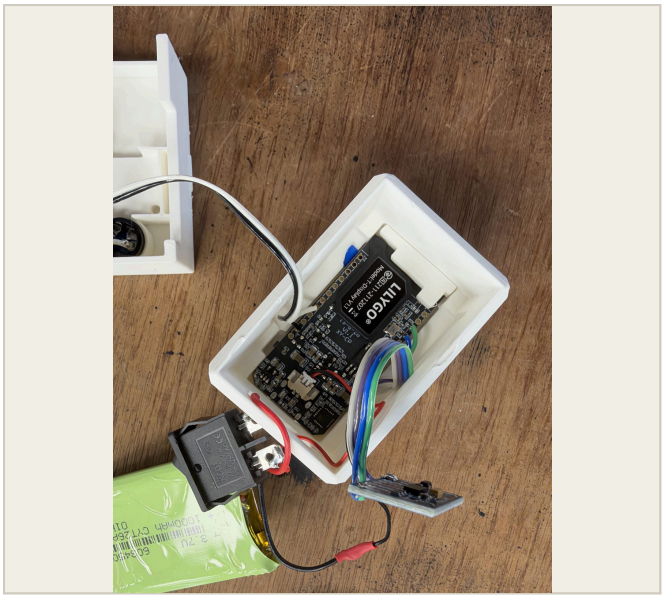
MEMS 加速度計原理示意：一塊微小的質量塊用彈簧吊在框架裡，加速時質量塊相對框架位移，位移量就對應加速度。實際晶片裡的「彈簧」是蝕刻出的摺疊矽樑，位移靠梳狀電極的電容變化量出來。（AI 生成示意圖，非實際晶片結構）

2.4　T-Display 上你會用到的腳位

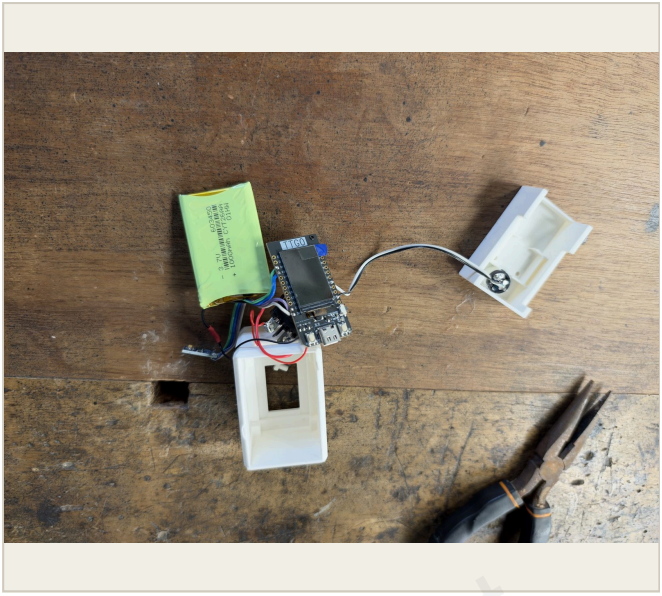
腳位	用途	要注意
GPIO0	下鍵	也是「開機模式」腳：開機瞬間按著會進入燒錄模式，所以不要讓程式在開機時依賴它
GPIO35	上鍵	只能輸入、**沒有內部上拉**，板子上有外接上拉電阻；自己接按鍵到 34～39 時要記得加電阻
GPIO21 / 22	I2C 的 SDA / SCL	ESP32 的 I2C 腳位可以指定，21 / 22 是 Arduino 預設；ADXL345 接這兩支
GPIO25	蜂鳴器	任何可輸出的 GPIO 都行；25 同時是 DAC 腳，之後想播放聲音檔也方便
GPIO34	電池電壓	板上分壓 1/2，程式讀到的電壓要 ×2；只能輸入
GPIO14	ADC_EN（電池量測開關）	官方腳位圖：要先把 GPIO14 拉高，GPIO34 才量得到電池電壓。我們第一版漏了這支腳，看官方資料才補上（第 7 節）
GPIO4、5、16、18、19、23	板載螢幕（背光、CS、DC、SCLK、MOSI、RST）	已經被螢幕用掉，不要再接別的東西

趙珩宇　生活科技專題教材

2.5　原作的外殼與組裝



T-Display 放進外殼下半部的凹槽，USB-C 朝外；電池紅黑線接 JST 座。



攤開：主板、電池、開關、ADXL345、蜂鳴器與兩件式 3D 列印外殼。排線用四芯彩虹線，剪下後直接焊在板子的通孔上。

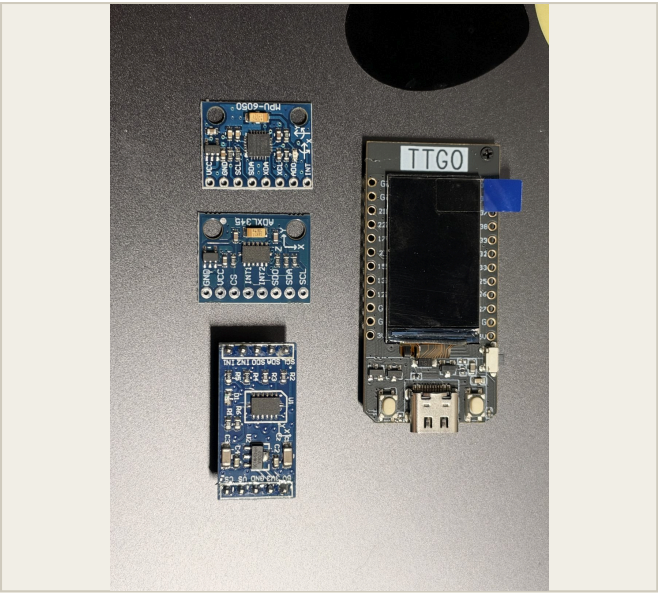
是什麼　兩件式 3D 列印外殼：下半部放主板、電池與開關，上蓋放感測器與蜂鳴器；感測器用四芯排線接到主板。

為什麼　感測器放上蓋、固定在殼上，走路時的震動才會完整傳到晶片；放在鬆動的電池旁會多出很多雜訊。蜂鳴器靠近蓋子出音孔聲音才大。

所以呢　排線長度要留夠開蓋，但不能壓到電池邊緣（鋰電池被尖物壓到有危險）。原作開關的裸露焊點是風險點；新版不裝開關，原本放開關的位置空下來，電池紅黑線直接插 JST。新版韌體不改任何接線，直接燒進這個殼就能用。

2.6　實作：剝線、沾錫、焊接（新版實作紀錄，2026-10-07）

新版的基礎組合只有四樣：ADXL345、無源蜂鳴器、鋰電池（直接插 JST）、主板。接線總共 6 條：ADXL345 四條（3V3 / GND / SDA / SCL）、蜂鳴器兩條（GPIO25 / GND）。下面是實際做一遍的紀錄。



手邊的三塊感測板與主板。上：MPU-6050（GY-521，8 腳）；中：ADXL345（GY-291，8 腳橫排）；下：另一款 ADXL345 模組（腳位分兩排，**同時有 5V 與 3V3 腳**，接哪一支都可以，但 I2C 仍是 3.3 V）。腳位名稱相同，接法一樣。



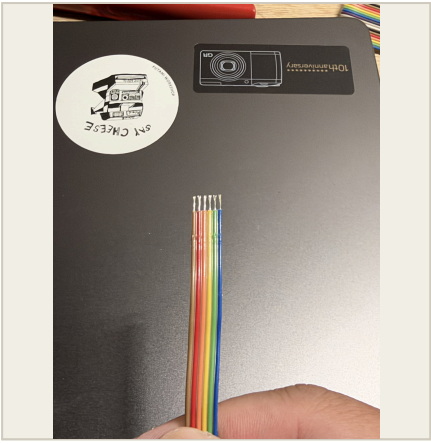
自動剝線鉗與彩虹排線。排線先撕成四芯一組（感測器）、兩芯一組（蜂鳴器），長度約 10 cm，留開蓋的餘裕。



圖 2-2　每一條線都走同樣三步：剝 3 mm → 線頭沾錫 → 插孔焊接再剪齊。順序是蜂鳴器 → ADXL345 → 主板，每焊完一個模組就用電表量 VCC—GND 有沒有短路。



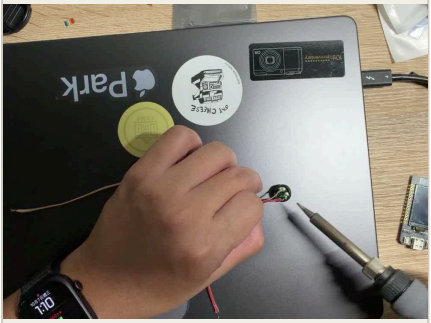
① 剝線：單線放進剝線鉗齒口，一夾一拉，絕緣皮就脫開。



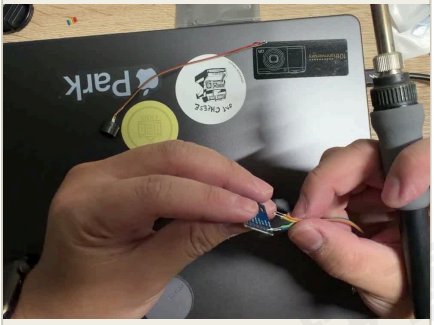
四芯都剝好、長度一致（約 3 mm）。



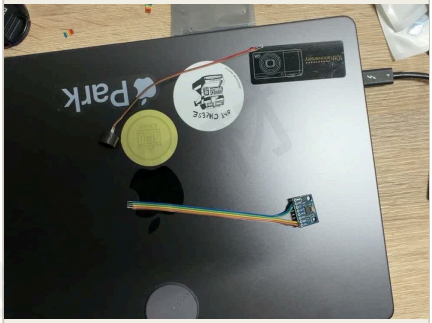
② 沾錫後線頭變成硬棒，用手指確認沒有散開的銅絲。



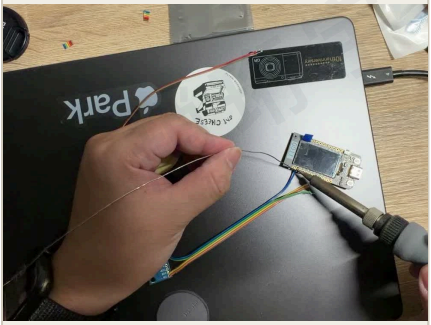
③ 先焊最簡單的蜂鳴器：兩支腳各一條線，烙鐵同時碰腳與線。



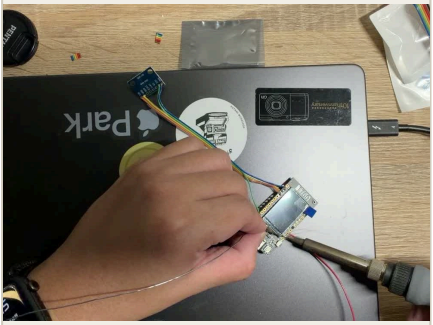
④ ADXL345：四條線穿過 VCC / GND / SDA / SCL 的通孔從背面焊，其他四孔空著。



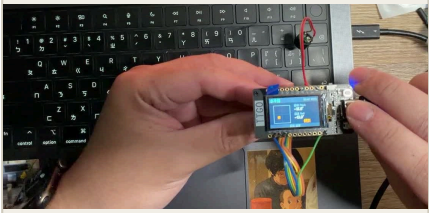
模組端完成，排線顏色對應腳位要記下來（例如 綠 SDA、藍 SCL）。



⑤ 主板端：3V、G、21、22 與 25、G 共六孔，排針孔比模組大，容易焊。



⑥ 全部焊好：主板—排線—ADXL345，蜂鳴器另一組兩芯線。



⑦ 插 USB 開機，序列埠印出「[ADXL345] 初始化成功」，螢幕進時鐘畫面。

做完的驗收

序列埠開機訊息看到 `[ADXL345] 初始化成功`；`status` 指令回傳 `accel_ok: true`，俯仰 / 橫滾值會跟著手上的動作變；按任一鍵有「嗶」聲。這一組接上之後，計步、擲筊、骰子全部啟用。

實作時的提醒

① 焊接期間模組還沒裝進殼，排線很容易在焊點處折到——焊完先套熱縮管再整線。② 模組建議平貼上蓋、晶片面朝上。每個人裝的方向難免不同——計步只看三軸合成的加速度，不受方向影響；但任何要用到「哪邊朝下」的功能都會因人而異，這也是 v1.9.4 拿掉水平儀的原因。③ 焊主板時排線不要蓋住 USB-C 與按鍵。

趙珩宇　生活科技專題教材

3　接線與原因：為什麼接這幾支腳

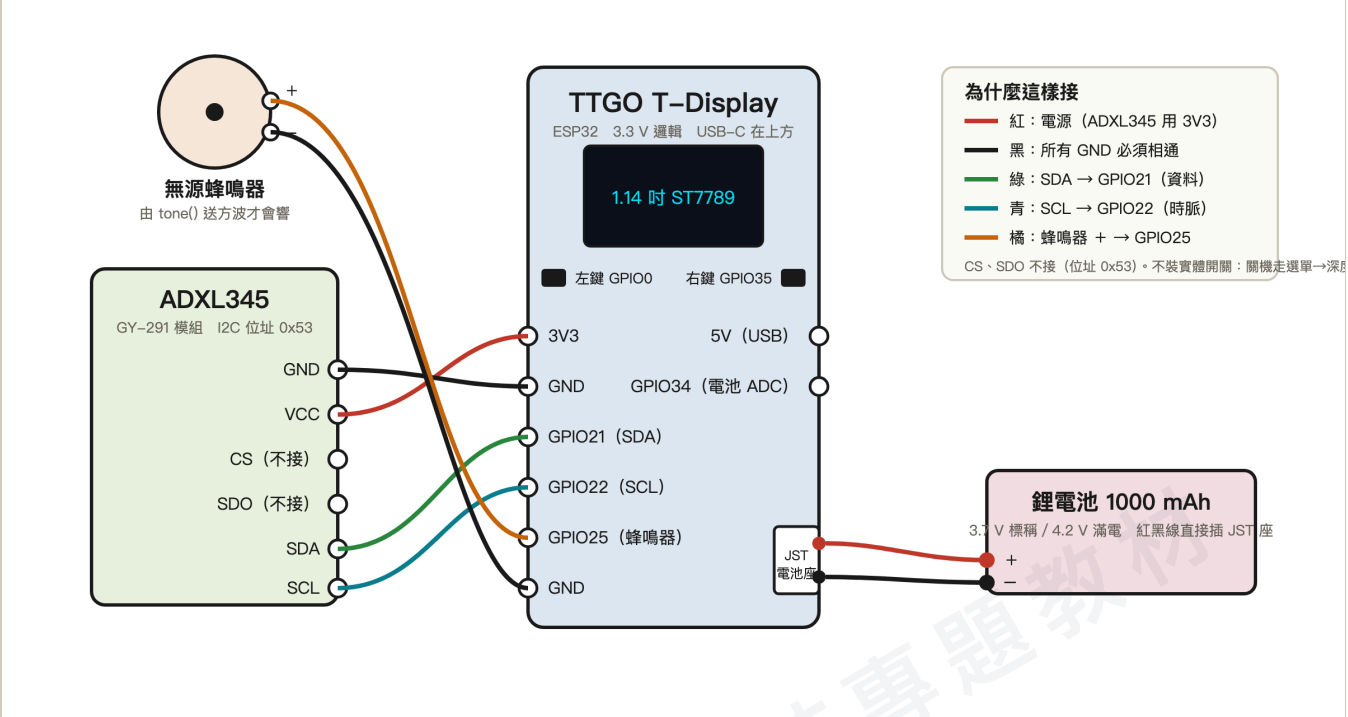


圖 3–1　完整接線（新版）：ADXL345 四條、蜂鳴器兩條、電池紅黑線直接插 JST 座，沒有實體開關。顏色對應右上角圖例；CS、SDO 兩支腳不接。

3.1　I2C 只要兩條線

是什麼　I2C 是一種「一條資料線（SDA）＋一條時脈線（SCL）」的匯流排，同一對線上可以掛很多顆裝置，靠**位址**區分。

為什麼　兩條線都是「開汲極」：裝置只會把線拉低，放手時要靠**上拉電阻**把線拉回高電位。GY–291 模組板上已經有上拉電阻，所以我們不用再加。

所以呢　SDA、SCL **接反就完全不通**（程式會報「找不到加速度計」）；位址對不上也不通。ADXL345 的位址由 SDO 腳決定：接地是 0x53、接高是 0x1D，模組板已把 SDO 拉低，所以不接就是 0x53，程式也是用 0x53。

3.2　為什麼 ADXL345 的電源接 3V3？

ADXL345 晶片本身只能吃 2.0～3.6 V。GY–291 模組上有一顆 3.3 V 穩壓器，所以接 5 V 也能動，但 SDA / SCL 的高電位仍是 3.3 V，直接接 3V3 最單純，也省掉穩壓器的壓差損耗。這也是為什麼在挑模組時要看「板上有沒有穩壓器」——沒有的裸晶片板接 5 V 會直接燒掉。

趙哲毛 生活科技專題教材

3.3　共地

電壓是「相對」的：ESP32 判定輸入為高電位的門檻是 0.75×VDD（約 2.5 V）、低電位是 0.25×VDD（約 0.8 V）——都是以**板子自己的 GND** 為 0 V 量的。感測器若沒和板子共地，兩邊的 0 V 會各自漂移，I2C 的高低準位就判不準、通訊直接中斷。所有零件的 GND 都要連在一起，電壓才有共同的參考點。模擬器的「接線」分頁第一條規則就是檢查這件事：蜂鳴器的負極、ADXL345 的 GND、電池負極，少接任何一條都會被標出來。

3.4　電池電壓怎麼量：GPIO34＋GPIO14（ADC_EN）

是什麼　鋰電池最高 4.2 V，超過 ADC 能量的範圍，板上用兩顆 100 kΩ 分壓成一半接到 GPIO34，程式讀到的值要 ×2。

為什麼　分壓電阻若永遠接著電池，會一直漏約 21 μA（4.2 V ÷ 200 kΩ）。原廠在這條路上加了一顆 MOSFET，由 **GPIO14（ADC_EN）** 控制：拉高才導通、才量得到。這就是我們第一版「電量亂跳」的原因。

所以呢　分壓節點上還有一顆 1 μF 電容（電路圖 C14），時間常數約 50 ms——「量前才拉高、等 200 μs 就讀」會讀到 0 V（我們第一版就是這樣，拔掉 USB 電量永遠 0%）。最後照官方範例把 GPIO14 常開（多耗約 21 μA），只在深度睡眠前拉低。ESP32 用 `analogReadMilliVolts()` 讀回已校正的毫伏值，每次取 4 筆平均。

顯示：10 格電池圖示，不顯示跳動的數字

電壓每秒讀一次、經指數平滑（係數 0.15），換算成 0～10 格；**跨格要超過 4% 的遲滯**才換格，所以不會在 79 / 80 之間抖。插著 USB 時量測點讀到的是 USB 的 4.9 V 而不是電池，程式以「> 4.35 V」判定為充電中：圖示上畫閃電、格數沿用拔 USB 前最後一次（存在 NVS，重開機還在）。這跟手機電量顯示的做法相同——使用者要的是「大概還剩多少」，不是毫伏。



時鐘右上角：10 格電池圖示（綠 / 橘 / 紅三段色）＋百分比；充電時加閃電。

動手前先在模擬器接一次

模擬器的「接線」分頁可以拖線、按「檢查接線」，九條規則會逐條回饋（短路、GPIO 碰電源、EN 接地、共地、I2C 接反、位址、CS 接地切成 SPI、蜂鳴器腳位、電池極性）。接錯會真的反映到模擬：SDA / SCL 接反重開機後螢幕就出現「找不到加速度計」。



圖 3–2　模擬器的接線關卡：示範接法檢查全部通過；SDO 不接會以「△」提醒原因，不算錯。

趙珩宇　生活科技趣教材

4　從加速度到步數：演算法怎麼想

先看效果，再講參數。下圖是把**韌體裡那份真正的演算法**餵進合成訊號後倒出來的中間結果：前 2 秒靜止、2～10 秒以每分鐘 110 步走路、12.5 秒時搖晃一下。

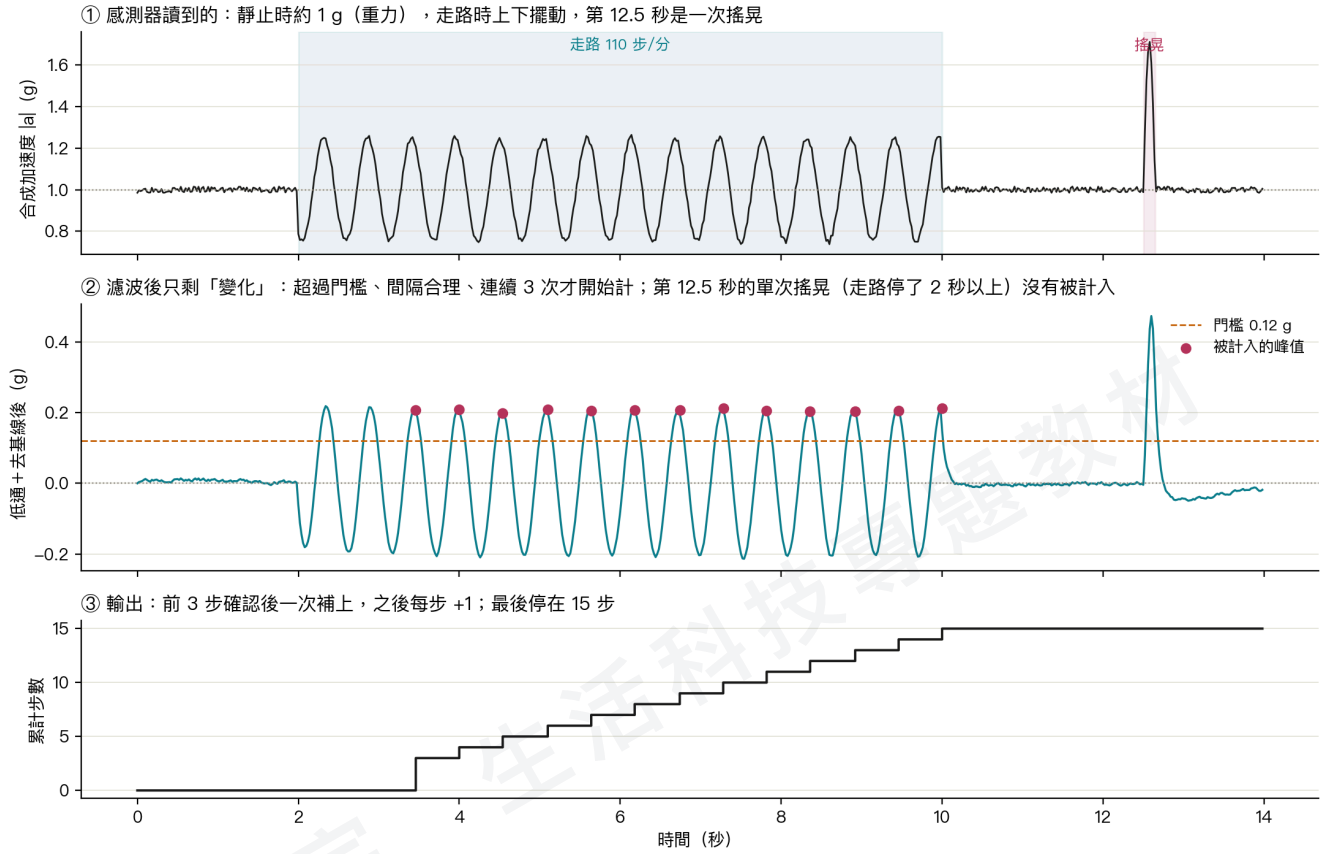


圖 4–1　上：感測器讀到的合成加速度，靜止時是 1 g。中：濾掉重力與慢變化後只剩「變化」，橘色虛線是門檻，粉紅點是被計入的峰值。下：累計步數。

4.1　三道處理

1. **合成加速度**　`|a| = √(ax²+ay²+az²)`。用合成值而不是某一軸，裝置怎麼戴、怎麼轉都不影響。
2. **低通＋去基線**　先用低通濾掉高頻抖動（`lp += 0.3·(|a|−lp)`），再減掉緩慢變化的基線（`dc += 0.02·(lp−dc)`）。基線就是重力加上慢慢變化的姿勢，剩下的才是「每一步」。

   數學上這兩個都是一階 IIR 低通 `y[k] = y[k−1] + α·(x[k] − y[k−1])`，截止頻率 $f_c = -\ln(1-\alpha) / (2\pi \cdot \Delta t)$。取樣 20 ms 時：α=0.3 → $f_c \approx 2.8$ Hz（走路 1.5～3 步/秒剛好留著，腳跟落地的尖銳衝擊被濾掉）；α=0.02 → $f_c \approx 0.16$ Hz、時間常數約 1 秒（只跟得上姿勢的慢變化）。
3. **峰值 → 間隔 → 連續確認**　訊號由升轉降且高過門檻（0.12 g）算一個候選步；和上一步間隔要在「最短間隔」～2000 ms 之間（太快是同一步的二次震動、太久是停下來）；連續 3 個合格候選才開始計，之後每個峰值 +1。

超新星 生活科技專題教材

第一版的最短間隔固定 250 ms，放在褲子口袋實測會多算 13～29%。現行 v4 加了「重新上膛」，最短間隔也改成由自相關估出的步伐週期自動決定，詳見第 14 節。

4.2　四個可調參數

參數	預設	調大會怎樣	調小會怎樣
峰值門檻 k_thresh	0.12 g	輕步不算、搖晃也不算	敏感，桌子震一下就計
最短間隔下限	300 ms（v1 是 250）	快跑（>200 步/分）會漏。v4 實際用 max(300, 0.7×步伐週期)	一步內的二次震動被算成兩步
最長間隔	2000 ms	慢走較不會中斷，但停下後的一次碰撞也可能被接續計入	慢走會一直重新「連續確認」，前幾步被吃掉
連續確認	3 步	更不會誤判，起步前幾步顯示延遲	反應快，單次搖晃容易被計

一個誠實的限制

連續三次間隔 0.7 秒的搖晃，對這個演算法來說就是「走路」——它和真的走路在訊號上沒有差別。手環用的進階做法會再看頻率是否穩定、三軸能量分布，或是直接用晶片內建的計步器（第 5 節）。這正是教學上值得討論的點：演算法在「規則」與「例外」間取捨。

4.3　實驗：在模擬器裡調參數

1. 打開模擬器 → 同按短放進入「計步」→ 按「開始走路」，看步數是否在 3 秒後開始跳、每分鐘約 110 步。
2. 停止走路，按一下「搖晃一下」，等 3 秒：步數不應該變。再快速連按三下：步數會 +3——想想為什麼。
3. 切到「儀表板 → 設定」，把門檻改成 0.05，再走路與搖晃，觀察波形圖的橘色虛線與誤判。改回 0.12，再試 0.3（這時振幅 0.25 g 的走路會完全計不到）。
4. 把「振幅」滑桿拉到 0.08 g（像是把裝置放在口袋深處輕走），看哪個門檻才計得到；討論「敏感」與「誤判」的取捨。
5. 燒進真機後，用手機連熱點開儀表板，實際走 100 步比對誤差，再回頭調參數。

4.4　距離與卡路里是怎麼算的

距離 ＝ 步數 × 步幅；活動消耗 ≈ 距離（km）× 體重（kg）× 1.036。這是常見的步行概估公式，誤差不小——步幅會隨速度改變、坡度沒有被考慮。把這兩個數字看成「參考」而不是「量測」，是使用任何計步器都該有的素養。

新版把「個人資料」做進設定選單與網頁：性別、年齡、身高、體重。**步幅**預設由身高推算（男 0.415×身高、女 0.413×身高，常見的經驗係數），也可手動輸入；**基礎代謝**用 Mifflin–St Jeor 算出來顯示在裝置與網頁；**今日總消耗估計**＝活動消耗＋基礎代謝×今日已過的時間比例。這三個數字分開顯示，學生才看得出「手環的數字為什麼比較大」是因為加了基礎代謝。

> **為什麼手環顯示的卡路里比我們多很多？**
>
> 本裝置算的是**活動消耗**（走路額外燒掉的）。市售手環顯示的「總消耗」還加上**基礎代謝率**（BMR，躺著不動也在燒的能量），例如 Mifflin–St Jeor 公式：男 BMR ≈ 10×體重 + 6.25×身高 − 5×年齡 + 5（女 −161）kcal/天，60 kg、170 cm、16 歲男生約 1,620 kcal/天，平均每小時 67 kcal——光這一項就比走 1 小時的活動消耗還多。不先分清楚這條邊界，學生會以為自己的演算法少算了。

4.5　傾斜角：同一顆感測器的另一種用法（延伸題）

裝置上的水平儀畫面在 v1.9.4 拿掉了（每個人的感測器安裝方向不同，讀數不一致，見 13.6 節）；傾斜角的計算仍保留在程式裡，遊戲卡帶可以用 `GameCtx` 的 pitch、roll 做傾斜遊戲。

靜止時三軸量到的是重力向量，向量的方向就是「哪邊朝下」。俯仰角 `pitch = atan2(-ax, √(ay²+az²))`、橫滾角 `roll = atan2(ay, az)`，再做一點平滑。這說明了為什麼第 5 節選型時「只要加速度計」就夠：傾斜量測不需要陀螺儀——除非你要量「轉得多快」或在移動中量姿態。



裝置綁在腳踝、手腕或放口袋，訊號都不同：腳踝震動最強、手腕疊加手臂擺動、口袋最接近身體質心。實測時請固定一種戴法再比較參數。（AI 生成示意圖）

趙珩宇　生活科技專題教材

5　模組比較與選型：換一顆行不行

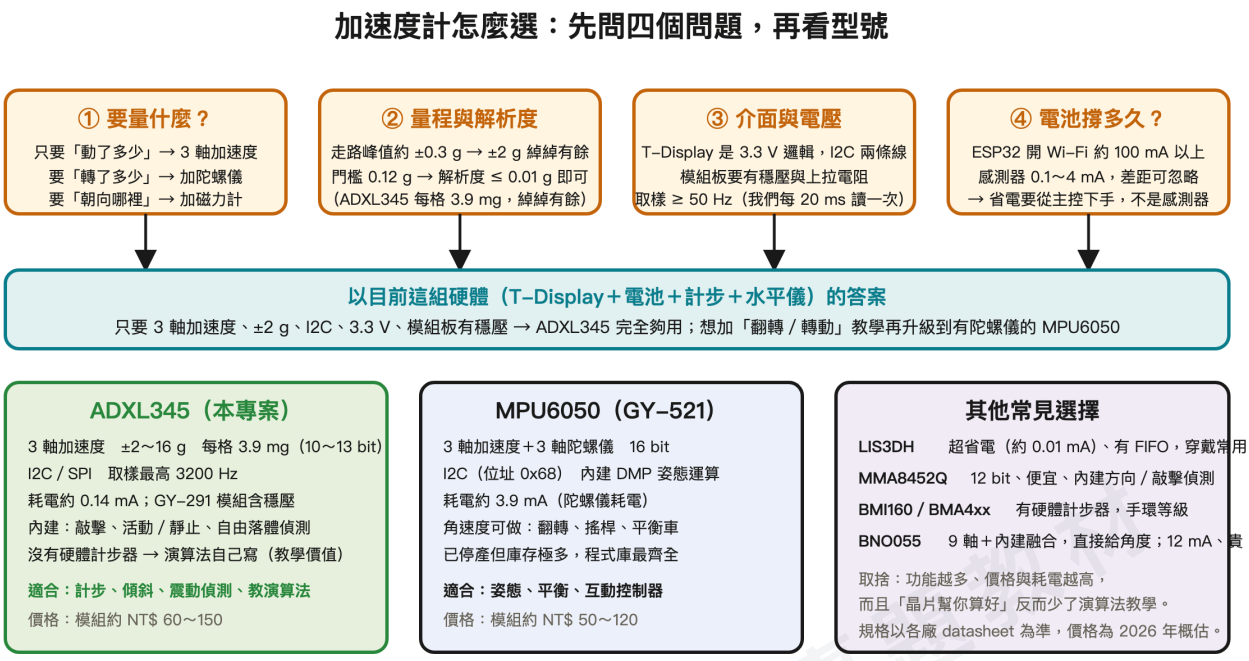


圖 5–1　選加速度計先問四個問題。對目前這組硬體而言，ADXL345 已經足夠；要教「轉動」再上 MPU6050。

5.1　加速度計 / 慣性感測器

型號	量什麼	介面 / 位址	解析度	耗電（量測中）	內建功能	對本專案的評價
ADXL345	3 軸加速度	I2C 0x53 / 0x1D、SPI	3.9 mg / 格	約 0.14 mA	敲擊、活動 / 靜止、自由落體	**夠用；演算法自己寫，教學價值最高**
MPU6050	加速度＋陀螺儀	I2C 0x68 / 0x69	16 bit	約 3.9 mA	DMP 姿態融合	多了轉動；已停產但庫存多、資料最齊
LIS3DH	3 軸加速度	I2C 0x18 / 0x19、SPI	12 bit	0.002～0.011 mA	FIFO、敲擊偵測	最省電；想做「電池撐一週」的版本再換
MMA8452Q	3 軸加速度	I2C 0x1C / 0x1D	12 bit	0.006～0.165 mA	方向、敲擊、自由落體	便宜替代；注意位址可能與 ADXL345 的 0x1D 撞
BMI160 / BMA456	加速度（＋陀螺儀）	I2C / SPI	16 bit	約 0.18 mA	**硬體計步器**	手環等級；晶片直接給步數，少了演算法教學

BMI270	加速度＋陀螺儀	I2C 0x68 / 0x69、SPI	16 bit	約 0.69 mA（雙感測）	**硬體計步器**、手勢、姿態辨識	現代穿戴主流（Bosch）；省電，但步數由晶片給，少了演算法教學
ICM-42688-P	加速度＋陀螺儀	I2C、SPI	16 bit	約 0.88 mA	低雜訊（加速度 70 μg/√Hz）、2 KB FIFO	無人機、工業級；對計步是殺雞用牛刀
BNO055	9 軸＋融合	I2C 0x28	—	約 12 mA	直接輸出角度、四元數	貴、耗電；適合機器人姿態，不適合計步器

規格取自各廠資料手冊的典型值，不同模組板（有無穩壓、上拉）會影響實際接法與耗電；價格（台灣零售，2026 年）：ADXL345 模組約 NT$ 60～150、GY-521 約 NT$ 50～120、LIS3DH 模組約 NT$ 100～250、BNO055 模組約 NT$ 600～1200。

5.2 選擇條件（針對目前這組硬體）

條件	這組硬體的需求	判斷
量測種類	計步只要「動了多少」；水平儀只要重力方向	3 軸加速度即可，陀螺儀是加分不是必要
量程	走路峰值約 ±0.3 g、跑步約 ±1 g	±2 g 檔位足夠；留 ±4 g 給跑步
解析度	門檻 0.12 g，要能分辨 0.01 g 等級的變化	3.9 mg / 格綽綽有餘
取樣率	程式每 20 ms 讀一次（50 Hz）	所有候選都遠超過；ADXL345 設 100 Hz 頻寬
介面與電壓	T-Display 是 3.3 V 邏輯，I2C 預設 21 / 22	模組板要有穩壓與上拉；避免位址互撞
耗電	ESP32 本身 80～240 mA	感測器 0.1～4 mA 都可忽略；省電要靠關 Wi-Fi、降頻、睡眠
程式庫與資料	學生要能查得到範例	ADXL345、MPU6050 資料最多；本專案直接讀暫存器，不依賴外部函式庫
教學目標	要讓學生「看到演算法」	選**沒有**硬體計步器的型號，才有東西可教

5.3 顯示器

方案	規格	優點	代價	適合
T-Display 板載 ST7789	1.14 吋、135×240、彩色、SPI	不用接線、彩色、速度快	佔掉 6 支 GPIO；中文字型要自己想辦法（第 7 節）	本專案

超級享 生活科技專題教材

SSD1306 OLED	0.96 吋、128×64、單色、I2C	兩條線、省電、便宜	小、單色；中文一行只放 8 個字	只顯示數字的簡化版
無螢幕	只靠蜂鳴器與網頁	最省電、最便宜	沒有手機就看不到	純感測記錄器
電子紙 2.13 吋	250×122、SPI	陽光下可讀、不耗電	更新慢（秒級）、不能動畫	一天看幾次的步數牌

5.4 主控板

板子	核心	無線	電池	程式語言	對本專案
TTGO T-Display	ESP32 雙核 240 MHz、4 MB	Wi-Fi、藍牙	有充電與電池座	Arduino C++ / MicroPython	**一片板子解決螢幕、按鍵、充電、網頁**
Raspberry Pi Pico W	RP2040 雙核 133 MHz、2 MB	Wi-Fi	無充電電路	MicroPython / C++	要另接螢幕與充電模組；MicroPython 上手快
ESP32-C3 SuperMini	單核 160 MHz、4 MB	Wi-Fi、BLE	無	Arduino / MicroPython	最小最便宜；要另接螢幕
Arduino Nano（AVR）	8 位元 16 MHz、32 KB	無	無	Arduino C++	放不下中文網頁與 Wi-Fi，不適合

5.5 電源

是什麼　1000 mAh 鋰電池。以 Wi-Fi 開著平均 120 mA 估，約 8 小時；若改成「只有開網頁時才開 Wi-Fi」可以撐一整天以上。

為什麼　耗電大戶是 ESP32 的無線電，不是感測器也不是螢幕背光（背光約 20 mA）。所以省電的第一步是管理 Wi-Fi，而不是換更省電的加速度計。

所以呢　鋰電池不要放電到 3.3 V 以下、不要刺穿或壓到、充電時不要離開；程式裡電量 0% 設在 3.3 V 就是這個原因。開關串在正極線上，隨時能徹底斷電。

趙珩宇 生活科技專題教材

給這組硬體的結論

加速度計維持 ADXL345；蜂鳴器維持無源（才能放旋律）；螢幕用板載的；電池 1000 mAh 夠一天教學使用。 真正值得升級的是**軟體**：加「閒置 5 分鐘自動關 Wi-Fi」能讓續航翻倍；想教轉動與姿態時，再把感測器換成 MPU6050——接線完全一樣（I2C 21 / 22），只改讀暫存器那一小段程式。

趙珩宇　生活科技專題教材

6　軟體架構與模擬器：先在電腦上跑

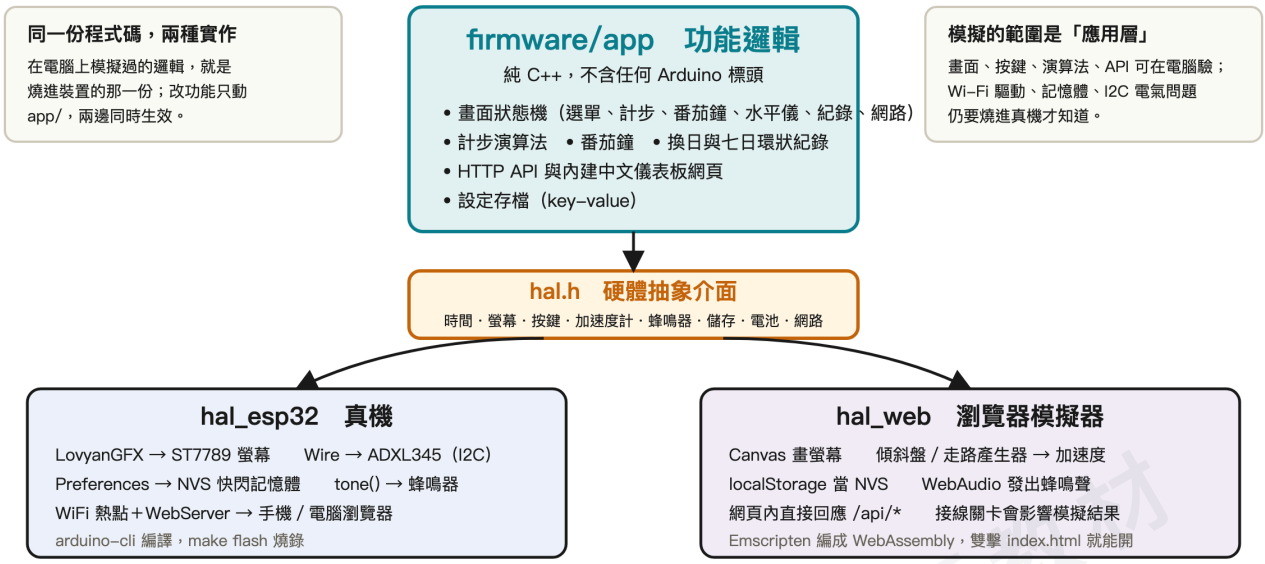


圖 6-1　功能邏輯只透過 hal.h 碰硬體，所以同一份程式既能燒進裝置，也能編成 WebAssembly 在瀏覽器執行。

6.1　為什麼要分層

是什麼　`firmware/app/` 是功能（畫面、計步、音樂、遊戲、API），`hal.h` 是一張「硬體能做什麼」的清單，`hal_esp32/` 與 `sim/hal_web/` 是這張清單的兩種實作。

為什麼　學生最常卡在「改一行要等兩分鐘燒錄、再盯著小螢幕看」。把功能和硬體分開，功能就能在電腦上以 600 倍速測，「閒置 2 分鐘自動關螢幕」只要 0.2 秒就測得到。

所以呢　模擬器通過不代表真機一定通過——Wi-Fi 驅動、記憶體不足、I2C 接觸不良都不在模擬範圍（第 13.4 節的螢幕翻轉、16.1 節的藍牙都是在真機才發現記憶體不夠）。模擬器負責「邏輯對不對」，真機負責「硬體行不行」。

趙玠元　崇左科技專題教材

6.2 模擬器介面



圖 6–2　左：裝置螢幕、兩顆按鍵、同按輔助鈕、模擬時間與速度。右：感測器（傾斜盤、走路產生器、波形）、序列埠、儲存（NVS）、儀表板、接線五個分頁。

分頁	可以做什麼	對應真機的什麼
感測器	拖圓盤改傾斜角；開始走路（步頻、振幅可調）；搖晃一下；看去重力後的波形與門檻線；電池電壓滑桿	ADXL345 的三軸讀值、GPIO34 電池電壓
序列埠	看程式印出的訊息（開機、存檔、換日、網頁操作）	USB 序列埠 115200 baud
儲存	看每個 key 的原始位元組與解讀；清除＝模擬抹除	Preferences / NVS 分區
儀表板	就是裝置內建的網頁，直接對模擬器的程式發 API	手機連熱點後看到的頁面
接線	拖線、檢查、示範正確接法；結果影響模擬	實際接線前的演練

6.3 建議的工作流程

1. **接線演練**：在「接線」分頁接一次、檢查、重開機，確認加速度計被偵測到。

鐵衛宇 生活科技專題教材

2. **功能走一遍**：主選單六項都進去看；用 600× 等自動關螢幕、再按「輕敲兩下」叫醒；「跳到明天」看七日紀錄換日。
3. **改程式**：只動 `firmware/app/`。`make sim-build` 幾秒後重新整理頁面。
4. **自動驗收**：`node tools/qc_sim.mjs`，目前 68 項機器判定（走路計步、單次搖晃不計、自動存檔、儀表板一致、音樂播放、自動關螢幕與輕敲叫醒、Wi-Fi 開關、藍牙畫面、換日、接線規則……）；`build/test_pedometer` 跑演算法的情境；`make test-walk` 用真機口袋錄音做回歸測試（14.5 節）。
5. **燒錄**：`make build`（會先自動產生字型）→ `make flash PORT=…`；或用瀏覽器的「寫程式 / 燒錄」頁、Wi-Fi 的 OTA（16.2 節）。燒錄腳本只接受允許清單裡的埠。
6. **真機驗證**：序列埠看開機訊息與 `perf`；插上電腦自動對時；用「走路測試」實走並輸入真值比對。

> **為什麼要有「機器可判定」的驗收**
>
> 「看起來正常」會漏掉很多事。這次 16 項驗收就抓到三個真正的錯誤：模擬迴圈被瀏覽器降速（30 秒只計到 4 步）、同按手勢用真實時間計時（加速時被判成長按）、重開機後時間倒退被當成換日。每一個都不是看畫面能發現的。

6.4　時間從哪裡來

ESP32 沒有電池時鐘，重開機就不知道今天幾號。新版有三個對時來源：① 插上電腦時裝置主動在序列埠問 `TIME?`，電腦端程式回答（13.1 節）；② 藍牙連線頁連上時寫入「對時」特性（16.1 節）；③ 手機或電腦打開儀表板，網頁把目前時間送給裝置（`POST /api/time`）。裝置記住「同步當下的開機毫秒數」，之後用差值推算。七日紀錄的「換日」只在推算日期**往後**跳時觸發——重開機後時間會暫時倒退，不能誤判成換日（這也是驗收抓到的錯之一）。狀態列顯示「時間未同步」就是在提醒使用者對一次時。用「關機」進深度睡眠時，時間會寫進 RTC 記憶體，開機後接回。

7　失敗路徑：我們踩過的坑

失敗不是附錄，是教材。下面每一條都是「做了什麼 → 哪裡不對 → 改成什麼 → 學到什麼」。

嘗試	失敗	改用	學到
用 LovyanGFX 內建的繁中字型（12、16、24 三個字級）	韌體 3.18 MB，程式分區只有 1.28 MB，連不上	建置時掃描程式用到的 271 個字，從系統字型描成點陣，只要 35 KB	「全字集」是為通用而設計；專案只需要自己用到的字。自動化（每次 make 重掃）才不會漏字
用 460800 baud 讀整片快閃記憶體	讀到 15% 斷線「Serial data stream stopped」	降到 115200，4 MB 約 6.5 分鐘，兩次讀取 SHA256 一致	速度與可靠度的取捨；備份一定要讀兩次比對
直接用現成的線上模擬器（Wokwi）	沒有 T-Display、沒有 ADXL345，裝置網頁要付費方案才連得到	自己做：同一份 C++ 編成 WebAssembly	工具不合需求時，評估「自己做」的成本；分層架構讓這件事變得可行
模擬迴圈用 requestAnimationFrame 驅動	瀏覽器在背景或無頭模式會把它壓到極低，30 秒走路只計到 4 步	改用 setInterval 驅動，rAF 只負責出畫面	先用原生單元測試證明演算法正確（55 步），才確定問題在驅動層——隔離問題
「同按短放」輔助鈕用真實時間 150 ms 放開	10× 加速時裝置端看到按了 1.5 秒，被判成長按回選單	按鍵持續時間改用模擬時間計算	時間有兩種：真實的與模擬的。所有「多久」都要問是哪一種
搖晃三下、間隔 0.7 秒，預期不計步	被計了 3 步	把測試改成單次搖晃；三下 0.7 秒本來就是走路步頻	測試寫錯和程式寫錯一樣常見；先想清楚「物理上該發生什麼」
重開機後沿用上次同步的時間	開機毫秒數歸零 → 推算日期倒退一天 → 觸發「換日」把今天清掉	只有日期往後跳才換日	邊界條件：時間不一定單調遞增

測深度睡眠時另開一個序列埠程式監聽喚醒	一開埠板子就重開機（rst:0x1 POWERON），測試結果看起來像「RTC 時間沒接回」	單一連線從同步、關機到喚醒跑到底（`tools/sleep_test.py`）	USB 轉序列晶片開埠會抖動 DTR / RTS，而那正是自動燒錄用的重置線；量測工具本身也會干擾被量的東西
燒錄到一半拔掉 USB	開機程式找不到可開的程式，一直重啟，螢幕全黑；看起來像板子壞了	插回 USB、看序列埠（會印 invalid segment length），重燒一次即可；NVS 的步數與設定都還在	燒錄中的 70 秒不要碰線；螢幕黑掉先看序列埠再下結論
電池量測前才拉高 GPIO14、等 200 µs 就讀	拔掉 USB 後電量永遠 0%	分壓節點有 1 µF 電容（電路圖 C14），時間常數約 50 ms；改成 GPIO14 常開（官方範例做法，多耗約 21 µA），關機才拉低	看電路圖不能只看電阻，電容決定「要等多久」
插 USB 時把量到的 4.9 V 當電池電壓	充電時永遠顯示 100%	> 4.35 V 判定為 USB 供電，顯示閃電並沿用上次格數	量測點在電源切換之後，先想清楚「量到的到底是誰」
電量顯示原始百分比	使用中數字一直跳（79↔80）	指數平滑＋10 格量化＋4% 遲滯	顯示給人看的數字要穩，精度留給 API
直接讀 GPIO34 當電池電壓	讀值不對：T-Display 的分壓電路要由 GPIO14（ADC_EN）啟用	查官方腳位圖，開機時把 GPIO14 拉高	板子的「特殊腳」要看原廠資料，不能只憑 Arduino 預設腳位想像
計步演算法用頻譜最高峰當步頻	1.74 Hz 是諧波，照它調出來的 v2 沒解決多算	問清楚真值怎麼數，再用自相關（第 14 節）	真值要問「怎麼數的」；頻譜最高那根不一定是你要的
螢幕翻轉配置 64.8 KB 暫存畫面	模擬器正常、真機最大連續記憶體只有 57 KB，不會轉	硬體旋轉＋動畫時用 8 位元 32 KB 暫存畫面；仍不穩定，v1.8.2 拿掉（13.4 節）	模擬器測不到資源限制
Bluedroid 藍牙	只剩 9.7 KB RAM，開機就當、重開機循環	先燒無藍牙版救回，改用 NimBLE（16.1 節）	可行性評估要同時量快閃記憶體和 RAM

趙珩宇 生活科技專題教材

perf 量測順便讀韌體大小	每量一次加速度計 FIFO 就溢位	慢的項目分到 `perf fs`（15.1 節）	量測工具也會干擾被量的東西
用 CRC32 比對畫面	12 ms，幾乎和送畫面一樣慢	旋轉 XOR 雜湊約 1 ms（15.2 節）	夠用的最簡單方法通常最好
網頁 API 設 Wi-Fi 完全關閉	只改數值，熱點仍開著	主迴圈統一套用（13.2 節）	同一設定多個入口要走同一段程式
原作韌體反組譯	見第 8 節	以行為規格重寫	見第 8 節

趙珩宇　生活科技專題教材

8　兩個逆向案例：Pico 成功、ESP32 停下

這學期我們有兩次「學生電腦壞了、原始碼不見了，只剩板子」的經驗。一次在 Raspberry Pi Pico 上**完整還原**了程式，這次在 ESP32 上**沒有**。差別不在運氣。

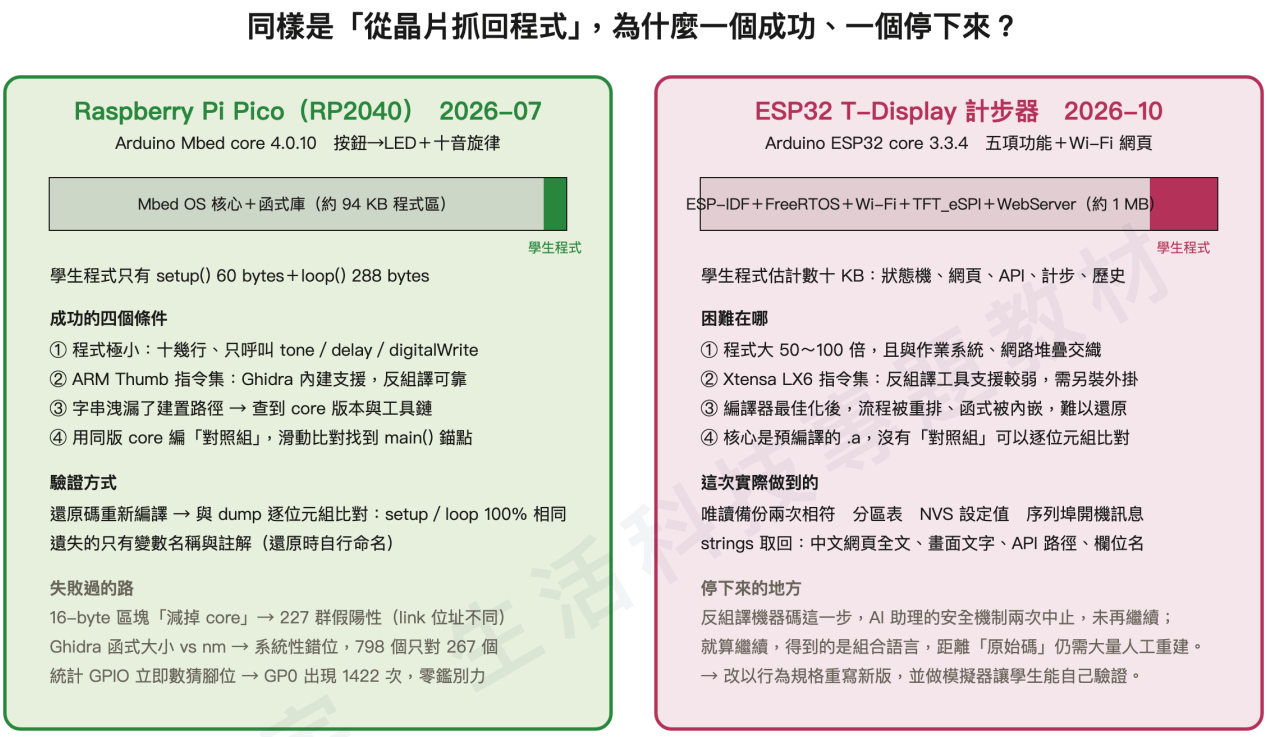


圖 8–1　兩個案例的對照。注意兩次都是 Arduino C++ 編譯後的機器碼，不是直譯式語言；Pico 那次成功是因為條件剛好全部成立。

8.1　先更正一個常見印象

很多人記得 Pico 那次是「抓回 Python 程式」。查了當時的紀錄（`pico-recover`），那片 Pico 上跑的是 **Arduino Mbed RP2040 核心**編譯的 C++，不是 MicroPython。還原是靠 `picotool save` 倒出 2 MB 快閃記憶體、Ghidra 反組譯、再用同版核心編譯對照組逐位元組比對，最後 `setup()` 與 `loop()` 100% 相符。

如果當時真的是 MicroPython，事情會更簡單：MicroPython 把 `main.py` 等原始碼**以明文存在板上的檔案系統**，用 Thonny 或 `mpremote fs cp` 直接複製出來就好，連反組譯都不用。這是直譯式語言在「可回收性」上的本質優勢——代價是執行速度與記憶體。

8.2 為什麼 Pico 可以

條件	Pico 那次	ESP32 這次
學生程式有多大	setup 60 bytes＋loop 288 bytes，十幾行	估計數十 KB：狀態機、內嵌網頁、API、計步、歷史
和系統的糾纏程度	只呼叫 tone / delay / digitalWrite，與核心界線清楚	FreeRTOS 多工、Wi-Fi 事件回呼、WebServer 處理函式散在各處
指令集與工具	ARM Thumb-2，Ghidra 內建支援成熟	Xtensa LX6，需另裝外掛，支援較弱
有沒有「對照組」	字串洩漏建置路徑 → 查到核心版本 → 同版編譯空白 / blink 程式，滑動比對找到 main() 錨點	核心是預編譯的 .a（arduino-lib-builder），版本查得到，但 1 MB 中要定位數十 KB 的使用者程式沒有可靠錨點
驗證方式	還原碼重新編譯 → 與映像逐位元組比對	無法做到同等強度的驗證
結果	原始碼 100% 等價（只失去變數名與註解）	取回明文的網頁與字串；機器碼反組譯這一步未進行

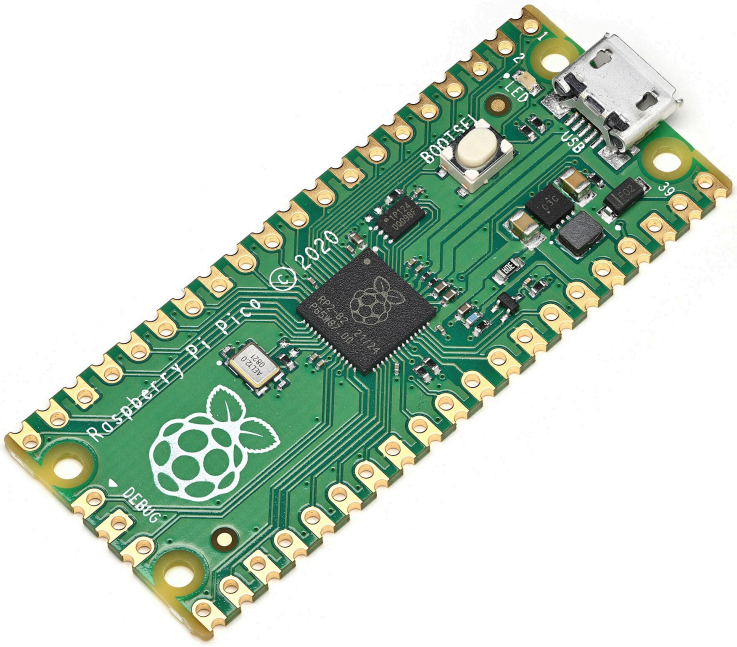
8.3 這次真正發生的事

唯讀備份、分區表、NVS 裡存的設定（步數 164、工作 6 分 / 休息 3 分）、序列埠開機訊息、以及用 `strings` 取回的整頁中文網頁與全部畫面文字，這些都拿到了。到了「把 Xtensa 機器碼反組譯回邏輯」這一步，協助分析的 AI 工具基於安全政策兩次中止，之後沒有再嘗試。即使繼續，得到的會是組合語言與偽碼，要還原成可讀的 C++ 仍需大量人工，而且無法像 Pico 那樣用「重新編譯逐位元組比對」來證明還原正確。

所以我們改走另一條路：把取回的畫面文字、API、資料欄位當成**行為規格**，重寫一份中文版，並做模擬器讓學生自己驗證行為。新版不是原作的複製品，而是以原作為規格的重新設計。

給學生的三個教訓

① 編譯後的機器碼不保留原始碼——變數名、註解、結構在編譯時就消失了。② 原始碼要放進版本控制（GitHub、GitLab）或至少雲端同步，「只存在一台電腦」等於沒有備份。③ 對不熟悉的板子，動手前先做**唯讀**備份並讀兩次比對；拖一個檔案進去就可能覆寫原程式（Pico 的 RPI-RP2 磁碟就是這樣）。



Raspberry Pi Pico（RP2040）。按住 BOOTSEL 插上電腦會變成一個磁碟，`picotool save -a` 可以唯讀倒出整片快閃記憶體。

趙珩宇 生活科技專題教材

9　延伸專案

六個由淺到深的延伸，完整程式與驗證方式都在模擬器網站的「延伸專案」頁（`sim/web/extensions.html`），可直接下載。

#	專案	做什麼、為什麼值得做	已做的驗證
A	**雙感測器比對與校正** 進階	ADXL345（0x53）與 MPU6050（0x68）掛同一條 I2C，貼在一起同步讀；兩顆各跑同一份計步演算法；按下鍵（GPIO0）做六面校正（每軸 ±1 g 求偏移與刻度）。電腦端腳本算靜止偏移、雜訊、波形相關係數、步數一致性並畫圖。回答「換一顆差多少」這個問題用數據，不用感覺。	arduino-cli 編譯通過；分析腳本可執行
B	**久坐提醒** 入門	原作說明書寫了但沒做：連續 60 分鐘沒步數就響鈴，之後每 15 分鐘再提醒，播放音樂時暫停。純 C++ 模組，三行接進 app.cpp；模擬器 600× 一分鐘就能看完一小時。	單元測試 9/9
C	**步頻穩定度** 挑戰	多要求最近三個步伐間隔的變異係數 ≤ 25% 才開始計。忽快忽慢的五下搖晃：原版計 5 步、穩定版 0 步；等距的三下搖晃兩版都擋不掉——因為它在物理上就是走路步頻。這個「擋不掉」本身就是討論題。	對照測試 6/6
D	**省電兩招** 進階	閒置 5 分鐘自動關 Wi-Fi、按鍵重開；長按下鍵（GPIO0）3 秒進深度睡眠（ESP32 約 10 μA）、上鍵（GPIO35）喚醒。這是 2.1 節第三種關機方式的實作，兼顧省電與充電。有電流表就量三種狀態。	arduino-cli 編譯通過
E	**電腦端後台** 進階	原作說明書的架構：裝置連家用 Wi-Fi、每分鐘 POST 步數到電腦，電腦存 CSV 並顯示全班總表。只用 Python 內建模組。與新版「裝置當伺服器」對照：幾個人用哪種、幾十台用哪種。	Python 可執行；裝置端編譯通過
G	**遊戲卡帶架構** 入門～挑戰	主選單多一項「遊戲」，每個小遊戲是一張卡帶（名稱 / begin / update / draw），計步在背景照跑。已內建擲筊、骰子、量角器三張；新卡帶只要繼承類別加一筆，模擬器直接驗。第 10 節詳述。	主韌體內建；模擬器驗收 5 項

趙珩宇　生活科技專題教材

F	**換成 MPU6050** 進階	主韌體已內建：編譯時加 `-DACCEL_MPU6050`（`make build-mpu`）就改讀 MPU6050。接線完全相同，只有讀暫存器那一小段不同——這就是硬體抽象層的價值。做完 A 再做 F 最順。	`make build-mpu` 編譯通過

討論題

1. 如果加速度計只給你 X 軸，計步會發生什麼事？裝置轉 90 度呢？
2. 手環的計步器幾乎都用「晶片內建計步」。對產品是好事，對學習呢？
3. 裝置沒有時鐘，我們靠網頁同步。還有哪些方法？各要付出什麼（成本、耗電、接線）？
4. 專案 C 擋不掉等距搖晃。還有什麼特徵能分辨「走路」和「有節奏地搖」？（提示：三軸能量分布、重力方向是否穩定）
5. Pico 的還原成功了，但花了一整天。如果當初多做哪一件事，這一天可以省下來？

專案 A 的做法（給要帶學生做的老師）

① 兩顆模組用雙面膠貼在同一塊小板上、軸向對齊，四線並接到 3V3 / GND / 21 / 22，MPU6050 的 AD0 不接。② `make ext NAME=dual_sensor_compare` 編譯、`make flash-ext` 燒錄。③ `python3 tools/compare_sensors.py /dev/cu.usbserial-XXXX --secs 60`：先靜置 10 秒、走 30 秒、再靜置；報告會列出兩顆的 |a| 平均（應為 1.000 g）、雜訊、相關係數、步數差。④ 按下鍵（GPIO0）進六面校正，再收一次，比較校正前後偏移是否收斂到 0.01 g 內。

趙珩宇 生活科技課題教材

10　遊戲化方向：讓學生想戴著走

這一節整理自另一場 AI 討論（Mac Studio 上的 Claude），參考專案連結已逐一查證存在；「卡帶架構」與第一批卡帶已實作進主韌體。

10.1　為什麼要遊戲化

計步器的目的就是「讓人多走」。只顯示數字很快就沒人看了；把步數變成遊戲資源、把走動變成遇見同學，動機才會留下來。而且遊戲是學生最有感的程式題——狀態機、亂數、計時、無線通訊都能順著做。

10.2　架構：主選單多一項「遊戲」，每個遊戲是一張「卡帶」

是什麼　`games.h` 定義 `Cartridge`：名稱、提示、`begin`、`update(GameCtx)`、`draw`。框架每圈把加速度、傾角、上下鍵事件、今日步數包成 `GameCtx` 餵給前景卡帶；兩鍵同按短放回卡帶列表、長按回主選單由框架處理。

為什麼　計步本來就在主迴圈背景跑，遊戲只是多一個前景畫面，所以**玩遊戲時步數照算**（模擬器驗收有這一項）。卡帶不碰硬體，所以寫完直接在模擬器驗，不用燒。

所以呢　新增一個遊戲＝新增一個類別、在 `CARTRIDGES` 陣列加一筆。第一批做了三張：**擲筊**（甩一下→聖 / 笑 / 陰筊，各配旋律）、**骰子**（1～3 顆）、**量角器**（傾角直讀、可歸零）；後來加了學生範例卡帶**步數目標**，v1.15 再加**貓咪抽籤**（連按上下鍵讓老師家那隻黑白貓伸長，搆到籤筒抽出今天的運勢籤，一天一支；靈感來自日本自造者「按鈕連打、貓會伸長」的作品），v1.7 依老師回饋移除量角器，v1.9.4 連水平儀也拿掉（13.6 節）。待機畫面有五種主題可在設定切換：小跑者（磚地、問號方塊、水管、金幣）、小恐龍（灰階沙漠、仙人掌、翼龍，每 100 步步數閃爍）、貓咪（夜晚屋頂、大樓剪影、星星與流星、小魚）、台鐵（小車站，普悠瑪、EMU3000、區間車、藍皮普快輪流進站停靠開出）、新幹線（富士山與高架橋，0 系、N700S、E5、E6、Doctor Yellow、500 系輪流）。角色與車輛都是原創點陣圖，車輛配色參考實車塗裝。v1.9 之前小恐龍與貓咪只換了角色、背景沿用小跑者的方塊與水管，老師看了指出「不能只換角色」，貓咪的頭尾也畫反了——這也是「實際使用者看了才發現」的例子。



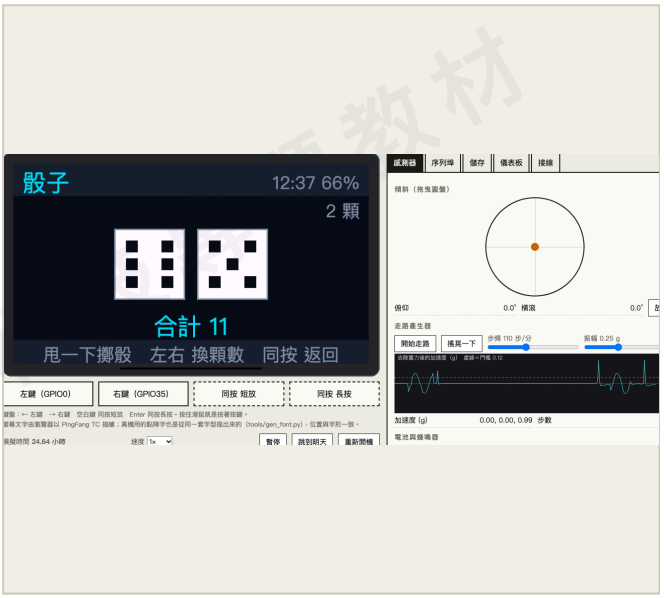
待機主題「小恐龍」：沙漠地平線、仙人掌障礙。



待機主題「貓咪」：屋瓦、煙囪、月亮。



擲筊：甩一下裝置，兩塊筊各自隨機平面或凸面朝上；一平一凸是聖筊（50%）、兩平是笑筊、兩凸是陰筊。亂數用加速度雜訊攪拌。



骰子：上下鍵換 1～3 顆，甩一下擲出。甩動偵測是「|a| 偏離 1 g 超過 0.6 g、靜止 350 ms 後才出結果」，避免一甩出好幾次。

順手拆解一個商品：Game & Watch: Super Mario Bros.（2020）的時鐘

它的時鐘畫面是「上面數字、下面一條關卡」：主角一直往右走、出畫面再從左邊進來，右邊會掉敵人，每整點有角色跳起來撞數字，特定時刻（例如 05:55）數字變金幣，一天藏了 35 個彩蛋，按鍵能換三種背景。 本版待機畫面借用這個**結構**——時刻在上、場景在下、以「時間事件」當彩蛋——但把彩蛋改成計步器的語言：**走路才會加速捲動、每 100 步噴金幣**。角色與場景都自己畫（12×16 像素圖），沒有使用任天堂的圖像。這也是一個好作業：挑一個喜歡的商品，拆出它的結構，再換成自己的內容。

10.3　A 路線：保留計步器，加遊戲性

玩法	怎麼做	用到的教學點 / 注意
賽博寵物	走路就是餵食與經驗值，累計步數達標進化；久坐一小時寵物變虛弱（接延伸專案 B）。參考 ESP32-TamaPetchi、Piko。	狀態機、NVS 存檔、像素動畫。`c.steps` 已在 GameCtx 裡。
擦身而過（StreetPass）	每台用 ESP-NOW 定時廣播自己的名片，別台經過時自動收下，集滿全班圖鑑；老師、別班算稀有卡。不需要校園 Wi-Fi。	無線通訊。**注意**：ESP-NOW 與 Wi-Fi 熱點 / 連線必須在**同一個頻道**（Espressif 文件）；要每隔幾秒才廣播一次，否則耗電。需在 hal.h 加廣播 / 接收介面。
步數探索 RPG	步數當體力，每走幾百步觸發一次遭遇，敲兩下裝置「捕捉」，收集品存 Flash。參考 picowalker。	ADXL345 內建敲擊偵測（TAP 暫存器）；亂數與機率。
全班打 Boss	延伸專案 E 的後台已有排行榜，再加一隻全班共用的 Boss，血量由全班總步數扣。	合作而非只比較；後端彙總。
成就徽章	連續 7 天達標、單日一萬步、凌晨還在走……	七日紀錄已有資料；只差判定與畫面。

10.4　B 路線：同一套硬體改成別的裝置

裝置	用到什麼	適合的教學點
擲筊 / 骰子 ✓ 已做	甩動偵測＋亂數	第一堂課就能完成；有在地文化梗
水平儀、量角器（做過，已移除）	傾斜角換算	工具類，很快做完；但每個人的感測器安裝方向不同，讀數不一致，v1.7、v1.9.4 先後拿掉
傾斜迷宮、傾斜打磚塊	三軸加速度→傾斜角→球的速度	X / Y / Z 軸與三角函數；參考 VolosR 的 T-Display 遊戲改成傾斜操控
西部決鬥（雙人）	ESP-NOW＋揮動偵測：嗶聲後先揮的人贏	無線通訊與時間同步
節奏遊戲	晶片內建敲擊偵測＋蜂鳴器節拍	中斷與計時
跑步配速器	蜂鳴器依目標步頻打拍，對照實際步頻	體育課可直接用（接體育班課程）

趣玩學生活科技專題教材

藍牙簡報翻頁器	模擬 BLE 鍵盤，傾斜或敲擊翻頁	老師自己用得到；BLE HID
跌倒、防盜警報	晶片內建自由落體與動靜偵測	長照或社會議題的設計題

為什麼不用藍牙來「先連線再帶去網頁」

想法很直覺：手機先用藍牙找到裝置，再自動加入它的 Wi-Fi 並開網頁。但三個地方卡住：① 手機作業系統不允許周邊裝置替手機加入網路（只有原生 App 能做，而且仍會跳確認）；② 瀏覽器用藍牙（Web Bluetooth）只有 Android Chrome 支援，iPhone 不行；③ ESP32 的藍牙堆疊要多 300～400 KB，目前韌體已占 89%，要換分區。 所以本版改用「螢幕 QR＋強制門戶」：不用 App、兩種手機都通、不增加 Flash。藍牙真正值得做的是「配網」——用 App 把家用 Wi-Fi 帳密傳給裝置（Espressif 有現成的 BLE Provisioning），讓裝置連上區網、用 mDNS 名稱開網頁，這留給延伸專案。

10.5　建議的開發順序與可借用的開源專案

1. **擲筊 / 骰子**（已做）：一堂課，先有成就感、也把卡帶架構學起來。
2. **賽博寵物＋擦身而過**：最能留住學生——寵物靠步數長大，也靠「認識朋友」解鎖造型。
3. **傾斜迷宮或西部決鬥**：給進度快的組別當挑戰。

類型	專案（已查證 2026-10）	可借用的部分
T-Display 遊戲	VolosR `SpaceWars`（T-Display 太空射擊）、`GameTTGOs3`（S3 版）；github.com/VolosR	直接跑得起來；把按鍵操控改成傾斜就接上感測器
電子雞	CyberXcyborg `ESP32-TamaPetchi`（有 LIS3DH 傾斜小遊戲）；hackaday.io `TamaFi`	寵物狀態機、成長與進化設計
計步＋寵物	Iloke-Alusala `ESP_Piko_Firmware`（MIT；LIS331 計步＋動作辨識動畫）；mamba2410 `picowalker`（RP2350 Pokéwalker 重製）	「步數換遊戲資源」、動作狀態（走 / 慢跑 / 衝刺）的門檻法
傾斜遊戲	sakshams7903-eng `The-Endless-Maze`（XIAO ESP32-S3＋OLED，非 T-Display）	傾斜控制、隨機產生迷宮

借用開源程式要看授權（上列多為 MIT / GPL），在教材與程式註解標明出處。

趙珩老師生活科技習題教材

11　專案歷程：從一台壞掉的電腦開始

這份講義本身就是一個案例：怎麼從沒有原始碼的裝置重建出一套可教學的系統。這一節把過程、決策與踩到的問題按時間整理，給想照著做的人看。

11.1　起點與目標

學生專題「GOTT Pedometer」做好了，但學生的電腦故障、原始碼遺失，手上只剩一台會動的裝置。目標有三：① 把裝置現狀完整備份，隨時能還原；② 盡量知道它在做什麼；③ 做出一個中文、可教學、學生能自己改的版本。

11.2　時間軸與關鍵決策

日期	做了什麼	關鍵決策與理由
10–04	唯讀備份整片 4 MB Flash（讀兩次 SHA256 一致）、解析分區表、NVS、序列埠開機訊息、字串	全程只用讀取指令；備份要讀兩次比對才算數。CH9102 用 460800 會斷線，降到 115200
10–04	嘗試反組譯機器碼	**停下**：Xtensa 工具支援弱、程式與 FreeRTOS / Wi-Fi 交織、無對照組可逐位元組驗證；對照 Pico 那次成功的四個條件這裡都不成立（第 8 節）。改以取回的明文（網頁、畫面字串、API、NVS 欄位）當**行為規格**
10–04	決定重寫而非修補；評估 Wokwi	Wokwi 沒有 T-Display / ADXL345、瀏覽器進裝置網頁要付費 → **自己做模擬器**。架構：功能邏輯純 C++，硬體抽象層兩種實作（真機 / Emscripten），同一份程式兩邊跑
10–04	第一版韌體編譯 3.18 MB 放不進 1.28 MB 分區	LovyanGFX 繁中全字集太大 → 寫 `gen_font.py` 建置時只描用到的字（35 KB）。不換分區，維持原廠配置
10–05	模擬器自動驗收腳本；燒入第二塊板	「看起來正常」不算驗收：16 項機器判定，抓到模擬迴圈被瀏覽器降速、同按手勢用錯時間基準、重開機時間倒退誤觸換日三個真錯誤
10–05	講義第一版；收到使用者實拍與原作說明書	發現**說明書與韌體漂移**（上傳後台 vs 自建熱點、久坐提醒沒做、開關接法不同）→ 寫成 1.1 節教學點
10–05	查官方腳位圖	發現電池量測要 GPIO14（ADC_EN）啟用——第一版漏了。教訓：板子的特殊腳要看原廠資料

10–06	遊戲卡帶架構、待機時鐘、Wi-Fi QR＋強制門戶、序列埠對時	另一場 AI 討論的遊戲化方向 → 做成「卡帶」介面；藍牙先連再帶去網頁不可行（OS 不許、iPhone 無 Web Bluetooth、Flash 不夠）→ 改 QR＋強制門戶
10–06	另一個 AI 的「修訂增補版」講義	當審稿意見逐條查證：採用 5 項、修正 4 處錯誤（VIH/VIL、ICM-42688 雜訊、濾波截止、AN-900 編號），記在附錄 H
10–07	電源設計定案：不裝開關、選單關機、RTC 續時；ADXL345＋蜂鳴器焊接	比較四種關機方式（2.1 節）；真機端到端驗證深度睡眠→按鍵喚醒→時間接回
10–07	設定選單、個人資料與卡路里、待機主題、電池圖示	電量 0% 的真因是分壓電容的時間常數；燒錄中拔線造成程式區損毀、重燒即復原
10–07	v1.1：USB 進時鐘、自動對時、Wi-Fi 預設關、亮度 50%、OTA、網頁燒錄頁	老師回報發燙 → 先處理最耗電的 Wi-Fi；換分區表為 min_spiffs 以容納 OTA
10–07	v1.2～v1.5：計步 v2→v4、走路測試、Wi-Fi 完全關閉、狀態列、長按回時鐘、螢幕翻轉	把諧波當步頻是錯的（14.3 節）；螢幕翻轉在真機記憶體不夠，改硬體旋轉
10–08	v1.6：效能量測與五項省電、NimBLE 藍牙、藍牙 QR、走路測試真值	先量再改；Bluedroid 讓板子重開機循環 → NimBLE（16.1 節）
10–08	v1.7：移除番茄鐘與量角器、新增音樂	使用回饋：用不到的功能就刪；曲目只用公眾領域
10–08	v1.8：主選單改成單項＋動畫插圖＋轉盤；音樂按鍵分成「停止時選歌、播放中調音量」	老師回饋：選單不必一次列那麼多項；播放中最常做的是調音量
10–08	v1.9：五種待機主題（含台鐵、新幹線）、拿掉水平儀	老師回饋：背景要隨主題換；每人感測器方向不同，水平儀讀數不一致
10–08	v1.10：時鐘依 24 小時變化光影（日出、正午、夕陽、夜晚）與天氣（晴、多雲、雨、雷雨、霧）	老師想到的點子；天氣不是預報，是依日期與時段算出的固定變化
10–08～09	v1.11～v1.12：移植到 T-Display S3（更大螢幕）：獨立的 320×170 版面、每小時步數長條圖、心率頁（MAX30102）	老師買了新板子；兩塊板共用功能程式，版面各自一份

10–09	v1.13：拿起來亮螢幕、加速度計自動偵測（ADXL345 / MPU6050）、給老師的燒錄包與網頁燒錄頁	省電又好用；要讓其他老師不用改設定就能燒
10–09	v1.14：S3 隨身碟匯出、S3 輕敲兩下	結束隨身碟要先把 USB 切回；輕敲門檻用實測數據定

11.3 AI 怎麼設計這套系統（可以借用的方法）

先備份　任何會寫入的動作之前，先把現狀完整備份並驗證可還原。這次的「燒錄到一半拔線」就是靠備份心態撐住的：壞掉的只是程式區，重燒即可。

規格從證據來　不猜。畫面文字、API 路徑、NVS 欄位、說明書、實物照片、官方電路圖，每一條都標來源；說明書與實作不一致時，**信實物與程式**。

架構先分層　功能邏輯不碰硬體、硬體抽象層很薄。好處是模擬器幾乎免費就有、換感測器只改一小段、學生改功能不會弄壞硬體層。

驗收要機器可判定　每個功能都寫成「預期 vs 實際」的檢查（走 30 秒計 40～70 步、搖一下不計、60 秒後 NVS 有值、換日後歷史有昨天……），改程式就全跑一次。目前 68 項。

小步快跑、每步可見　每個功能：寫→模擬器驗→燒真機→看序列埠。一次只改一個變數，出錯才找得到原因。

把失敗寫進教材　第 7 節的每一列都是真的踩過的坑。失敗的原因往往比成功的做法更有教學價值。

外部意見當審稿　另一個模型的產出不直接採用，逐條查證、採用的併入、錯的列表更正（附錄 H）。多模型交叉，但最後要有人（或一個模型）負責判斷。

11.4 還沒做完的事

- 計步 v4 目前只有「褲子口袋」的真機資料；拿在手上、綁手腕、放包包還要各錄幾段有真值的走路測試。
- 真機尚未實測：輕敲兩下叫醒、light sleep 中按鍵叫醒、電池續航的省電前後對照（`ps 0` / `ps 1` 各跑一次放電）。
- 延伸專案 A（MPU6050 比對）、GY-273 羅盤卡帶、VL53L0X 步幅校正的實機驗證。
- 七段教學短片（剪輯在 Mac Studio 進行，工單 `handson/clips_job.json`）。

12　逐步實作步驟

照這個順序做，每一步都有「怎麼知道做對了」。預計 2～3 節課。

#	步驟	做法	怎麼知道做對了
0	先在模擬器玩一遍	開 `sim/web/index.html`（或網站），主選單六項都進去；「接線」分頁拖線並按檢查	接線檢查全部通過；知道兩鍵手勢
1	裸板燒錄	只接 T-Display，USB-A 轉 C 線；`make flash PORT=…`（或用燒好的 .bin）	螢幕進時鐘畫面；序列埠印「ADXL345 警告：找不到」是正常的
2	對時	插著電腦執行 `python3 tools/sync_time.py --watch`（或開網頁燒錄頁），裝置會自動問時間；也可用藍牙連線頁	時鐘顯示正確時刻，不是「--:--」
3	剝線、沾錫	四芯排線＋兩芯各剝 3 mm、線頭沾錫（2.6 節、圖 2-2）	線頭是亮亮的硬棒，不散絲
4	焊蜂鳴器	兩支腳各一條線	電表量兩腳不短路
5	焊 ADXL345	VCC / GND / SDA / SCL 四孔，其他四孔空著；記下顏色	電表量 VCC–GND 不短路
6	焊主板	3V、G、21、22（感測器）與 25、G（蜂鳴器）	相鄰孔不相連；再量一次 3V3–GND
7	通電測試	插 USB；`status` 看 `accel_ok`	序列埠「[ADXL345] 初始化成功」；傾斜板子時 `status` 的 pitch / roll 會變；按鍵有嗶聲
8	接電池	紅黑線直接插 JST（不剪線、不裝開關）；拔 USB	右上角電池圖示有格數；插回 USB 出現閃電
9	走路測試	設定→走路測試→選放的位置→倒數後走，自己數步數（左右腳各算一步）→按鍵停止→輸入真值	誤差 < 10%；不對就把錄音 `dump` 出來用 14.1 節的方法分析
10	個人資料	設定→個人資料（或網頁）輸入性別、年齡、身高、體重	卡路里與基礎代謝數字合理
11	關機與開機	設定→關機→同按確認；按上鍵開機	螢幕全黑；開機後時間還是對的（RTC 續時）

趙珩宇 生活科技專題教材

12	裝進外殼	感測器平貼上蓋、晶片朝上；排線套熱縮管	走幾步計步會動；甩一下擲筊有結果
13	手機連線	設定→藍牙→開啟（會重開機一次）→手機掃畫面上的 QR	連線頁的步數跟裝置一樣，走一步就更新
14	延伸	第 9、10 節任選；改學生卡帶後用網頁燒錄頁編譯、燒錄	各專案自己的驗證方式

每一步只改一個變數

卡住時回到上一步能過的狀態，再只改一件事。螢幕黑了先看序列埠（附錄 G），不要急著拆線。

趙珩宇　生活科技專題教材

13　第二輪改良：讓裝置更好用

v1.1～v1.15（2026-10-07～10-12）。每一項都先說「為什麼要改」，再說「怎麼改」與「怎麼確認改對了」。第一次出現的專有名詞，詳細解釋都在附錄 I。



第二輪改良：用了才知道哪裡不好用（AI 生成示意圖）

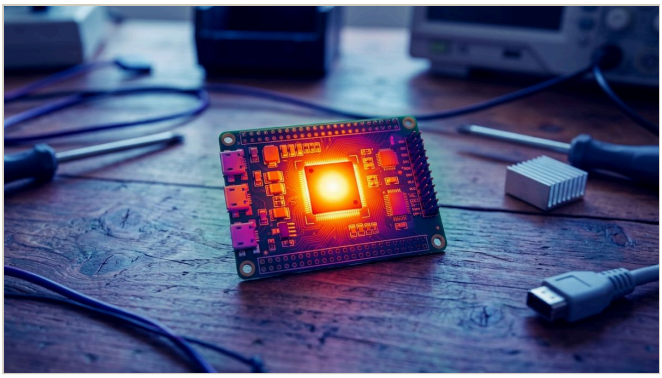
第一版做完、燒進板子之後，實際拿在手上用了兩天，冒出一連串問題：放在口袋步數跳很快、裝置會發燙、插著充電線也看不到時間、一定要開 Wi-Fi 才能用手機看資料。這一章是這些問題的總覽，後面三章再分別深入。

版本	改了什麼	為什麼
v1.1	接上 USB 直接進時鐘；接上電腦自動對時；Wi-Fi 預設關閉；亮度預設 50%；網頁 OTA 更新	老師回報「裝置運作中會發燙」，主因是 Wi-Fi 一直開著
v1.2～v1.5	計步演算法 v2→v4；「走路測試」畫面；`/api/live` 即時資料	老師回報「實際走路步數跳很快」
v1.3～v1.4	Wi-Fi 完全關閉開關；狀態列圖示；選單長按回時鐘；螢幕自動翻轉（v1.8.2 拿掉）	老師要求
v1.6	效能與省電五項：主迴圈休息、畫面沒變不送、加速度計 FIFO＋輕敲叫醒＋自動關螢幕＋light sleep、每分鐘步數紀錄、藍牙	實測發現 CPU 有 79% 的時間在空轉
v1.6.1	藍牙畫面加連線頁 QR 碼；走路測試可選位置、輸入實際步數、保留 5 段	為了讓手機好連、讓校正有真值

v1.7	移除番茄鐘與量角器；新增「音樂」	老師說平常用不到；想讓裝置能播歌
v1.8	主選單一次一項、動畫插圖、右側轉盤；音樂播放中上下鍵調音量；上下鍵對調（v1.8.1）	老師要求：畫面不必擠那麼多項；播放時按鍵要能調音量；上下鍵原本接反
v1.9	待機主題重做背景、新增台鐵與新幹線列車主題；貓咪改成老師家貓的黑白花色；拿掉水平儀（v1.9.4）	老師看了指出「不能只換角色」；水平儀讀數因安裝方向而異
v1.10	時鐘畫面依 24 小時變化：天空漸層、日月軌跡、夜裡星星與窗燈；天氣有晴、多雲、雨、雷雨、霧（依日期與時段的固定亂數）	老師想讓待機畫面「有時間感」。測試時也順便修掉「整點連續觸發時角色越跳越高」的舊問題
v1.11	延伸：移植到 LilyGO T-Display S3（1.9 吋 320×170、8 MB PSRAM）。功能程式共用，各頁版面每塊板子一份（`ui_240.h`、`ui_320.h`）；模擬器加 `?board=s3`	新板子螢幕大，置中加黑邊太浪費；老師要它有自己的排版
v1.12	延伸：S3 加 MPU6050 計步與 MAX30102 心率頁（手指量測、脈搏波形、每跳一聲）	新板子接了更多感測器。心率演算法第一版遇到心率變化會卡在舊值，被模擬器驗收抓到後修正
v1.13	拿起來亮螢幕（13.3 節）；加速度計 ADXL345 / MPU6050 開機自動偵測（15.3 節）；給其他老師的燒錄包與網頁燒錄頁（附錄 J）	省電又不用找按鍵；讓其他老師不必改設定就能燒兩種板子
v1.14	S3 插電腦變隨身碟匯出步數 CSV（16.2 節）；S3 輕敲兩下叫醒（13.3 節，門檻用三輪實測決定）	老師想直接在電腦上看資料；S3 的 MPU6050 沒有內建輕敲偵測
v1.15	遊戲卡帶「貓咪抽籤」：連按讓貓伸長抽運勢籤，一天一支（第 10 節）	老師看到日本作品想加入。模擬器驗收抓到：玩家還在連打就把剛抽到的籤按掉 → 結果出來 1.5 秒內的按鍵不算



第一版燒好後，老師實際拿在手上用了兩天（AI 生成示意圖）



「會發燙」：熱像儀看起來就像這樣，晶片附近最熱。本圖只是示意，沒有實際量溫度（AI 生成示意圖）

13.1 接上 USB 就進時鐘、接上電腦自動對時

是什麼　插上充電線（不管是插電腦還是插充電頭），畫面直接跳到待機時鐘；如果插的是電腦，裝置還會主動向電腦「要時間」。

怎麼做　裝置每秒量一次電池腳位的電壓。T-Display 的量測點在電源切換之後，插 USB 時量到的是 USB 的約 4.9 V，所以「超過 4.35 V」就判定為接上 USB。從電池變成 USB 的那一刻切到時鐘，接著在序列埠每 5 秒印一行 `TIME?`，最多問 30 秒。電腦端只要有程式回一行 `time 1791357911 480`（秒數與時區分鐘），裝置就對好時了。會回答的程式有三個：`tools/sync_time.py --watch`、網頁的「寫程式 / 燒錄」頁、模擬器。

所以呢　這是一種「裝置先開口問」的設計（拉，pull），比「電腦一直推時間過來」更省事：電腦端不必知道裝置什麼時候插上。

13.2 選單、狀態列與 Wi-Fi 完全關閉



主選單（v1.8）：一次一項＋動畫插圖，右邊轉盤顯示相鄰項目，換項時轉盤轉一格、內容從上或下滑進來。長按兩鍵直接回時鐘。



設定：無線網路、Wi-Fi 開關、藍牙、個人資料……共 12 項，右邊是目前的值。



Wi-Fi「完全關閉」時進無線網路只顯示說明；狀態列的訊號圖示多一條橘色斜線。

單項主選單（v1.8）：第一版的主選單一次列出四項文字，小螢幕上字擠、也看不出每項是做什麼的。改成一次只顯示一項：左邊 56×56 的小插圖會動（計步是左右腳印輪流往上走、音樂是跳動的等化器、七日紀錄是長條依序長出、遊戲是骰子點數變化、設定是三條滑桿），旁邊是大字項目名稱與一行即時資訊（例如今天的步數、正在播的歌）。右邊是一個**轉盤**：六個項目排在圓周上，目前的項目在正中間最大最亮，相鄰的項目在上下、越往後越小，轉到背面就不畫——就像從側面看一個

轉動的滾筒。換項時轉盤用 0.26 秒轉一格（先快後慢），這段時間畫面更新從每秒 10 張提高到約 33 張，轉完就降回來。真機實測停在主選單時 CPU 活動比例約 18%，和時鐘畫面差不多。

狀態列：每個畫面頂端都是「標題｜藍牙圖示｜Wi-Fi 圖示｜時間｜電池」。Wi-Fi 圖示：青色＝開著熱點、綠色＝連上家裡的網路、暗灰＝沒開、暗灰加橘色斜線＝完全關閉。藍牙圖示：藍色＝等待連線、亮藍＝已連線、沒開就不畫。電池圖示充電時會多一個閃電。待機時鐘畫面的數字佔掉中間 140 像素，所以電池、百分比、Wi-Fi、藍牙改成在右側直排、步數在左側分兩行（老師用手機拍到原本的圖示壓到時鐘數字，才發現要改）。

Wi-Fi 完全關閉：原本 Wi-Fi 是「進無線網路畫面才開，10 分鐘沒人用網頁就自動關」。再加一個「完全關閉」，不管怎麼按都不會開。開著 Wi-Fi 的 ESP32 約耗 100 mA 以上，是整塊板子最耗電、最熱的部分；而且這個版本的熱點畫面一開，CPU 也要從 80 MHz 升到 160 MHz。

> **失敗路徑：網頁設成「完全關閉」，熱點卻還開著**
>
> 嘗試：在設定 API 加 `wifi_en=0`。失敗：從裝置選單切換時會順手呼叫「關熱點」，但從網頁或序列埠改設定時只改了數值，正在開的熱點沒關。改用：設定變更後，下一圈主迴圈檢查到 `wifi_en=0` 就關熱點（要等網頁回應送出去之後再關，否則網頁會收不到「已儲存」）。學到：同一個設定有兩個以上的入口時，每個入口都要走同一段「套用」程式，不能各寫各的。

13.3 自動關螢幕與輕敲叫醒



「自動關螢幕」可選 1 / 2 / 5 分或不關（預設 2 分）。



螢幕關閉：背光 0、LCD 進睡眠。按任一鍵或輕敲外殼兩下叫醒，這一下只負責叫醒，不會誤觸選單。

螢幕背光是除了 Wi-Fi 之外最耗電的東西。沒按鍵超過設定時間就把背光關到 0、讓 LCD 控制器進睡眠模式（畫面內容保留，喚醒時不必重畫）。叫醒有兩種：按任一鍵，或**輕敲外殼兩下**。輕敲偵測是 ADXL345 晶片內建的功能（第 15.3 節），主程式只要每次讀資料時順便看一下它的狀態暫存器。因為走路時腳跟著地的衝擊可能被誤判成輕敲，所以規定「最近 2.5 秒內有算到步數」就不接受輕敲叫醒。

T-Display S3 的輕敲兩下（v1.14）：用數據定門檻

ADXL345 的輕敲偵測是晶片內建的；S3 用的 MPU6050 沒有，只能用程式從加速度裡找「很短、很尖」的變化。做法：MPU6050 改成每秒取樣 200 次（計步仍用每 4 筆平均成 50 次），看相鄰兩筆的變化量（jerk）；超過門檻算一下、70 ms 內的餘震算同一下，第二下在 80～600 ms 之內就算「輕敲兩下」。

> **失敗路徑：門檻用猜的**
>
> 嘗試：門檻先設 0.45 g。失敗：老師敲 6 下只算到 3 下。改用 0.25 g，又太敏感，誤算出 16 次。改用：讓裝置把每一下的大小與時間記下來（序列埠 `taplog`），請老師照「敲兩下 ×3 → 拿起來放下」做三輪：敲擊 1.0～3.4 g、一般晃動最大 0.71 g，門檻取中間 **0.85 g**；最後一輪剛好 3 次、放回桌上（單下 4.0 g）不算。學到：門檻要從實際資料的兩群（要抓的、不要抓的）之間找，不能憑感覺。

拿起來亮螢幕（v1.13）

老師想要像手錶一樣：螢幕關著時，**把裝置拿起來看就自動亮**，不必找按鍵。做法只用計步本來就在讀的加速度資料，所以不多花電：

步驟	規則	為什麼
1. 記住「放著的方向」	螢幕關掉後，裝置靜止 1.5 秒，就把當時的重力方向記成基準	每個人放的方式不同（平放、立著、倒放），不必校正，自動以「放下時」為準
2. 判斷「被拿起來」	重力方向和基準差 **35° 以上**，而且在新方向**穩住 0.4 秒**	只晃一下（碰到桌子）不會亮；要真的轉個角度拿著看
3. 排除走路	最近 2.5 秒內有算到步數就不判斷；停下來後要再靜止 1.5 秒才重新記基準	放在口袋走路時方向一直變，不排除的話會一直亮、更耗電
4. 沒人看就再關	拿起來亮的螢幕，15 秒內沒按鍵就自動再關；有按鍵就回到一般的自動關螢幕時間	拿起來只是看一眼時間

「設定 → 拿起來亮螢幕」可以關掉。怎麼確認做對了：模擬器加了 5 項自動驗收——螢幕關掉 → 平放靜止 → 改成立起來，螢幕要亮且回到時鐘；15 秒沒按鍵要再關；螢幕關著時走路**不能**亮；設定關掉後拿起來也不能亮。真機用序列埠 `status` 的 `raise_angle`（目前和基準差幾度）、`raise_wakes`（亮了幾次）、`raise_dbg`（最大角度、超過 35° 次數、沒拿穩、走路中被擋）觀察，`scroff` 可以馬上關螢幕來測。

失敗路徑：「拿起來沒反應」

嘗試：燒好後老師直接拿起來試。失敗：螢幕沒亮。查證：序列埠顯示偵測器其實正常，是**螢幕還沒自動關掉**（2 分鐘沒按鍵才關）就拿起來，這時本來就不會有變化；另外老師拿著不動超過 1.5 秒，偵測器會把「拿著的方向」當成新基準，角度歸零。改用：加 `scroff` 指令馬上關螢幕、加 `raise_dbg` 事後可讀的紀錄，照「放平數到 3 → 拿起來穩住 2 秒」測，實機確認會亮。學到：功能「沒反應」先確認前提條件有沒有成立，再懷疑程式。

夾角怎麼算

重力方向是一個三維向量 (x, y, z)。兩個方向各自除以長度變成「單位向量」後相乘相加（內積），結果就是夾角的 cos。cos 35° ≈ 0.819，內積小於 0.819 就代表轉超過 35°。不用算反三角函數，微控制器算起來很快。

13.4　螢幕自動翻轉：做了兩版，最後拿掉

老師希望倒過來拿時畫面跟著轉。v1.4 做了第一版、v1.5.1 修了一次，但老師實際拿在手上一直無法正常運作，v1.8.2 依老師要求整個拿掉，設定裡也不再有「螢幕方向」。下面保留當時的做法與兩次失敗，因為「什麼時候該停損」本身就是重要的一課。



（已移除）第一次打開「自動」要校正：把裝置正著立起來、螢幕對著自己，同按記住重力方向。



（已移除）模擬器裡倒過來拿約 0.4 秒，畫面用 0.35 秒的動畫轉 180°。

為什麼要校正　ADXL345 是用排線接的，模組在外殼裡擺什麼方向都有可能，晶片的 X / Y / Z 軸和螢幕的上下左右不一定對齊。所以先記下「正著拿時，重力在晶片座標裡指向哪裡」（一個長度為 1 的向量），之後只要算「現在的重力」和它的**內積**：接近 1＝正著、接近 −1＝倒著、接近 0＝平放（維持原方向）。

怎麼避免亂轉　重力先經過低通濾波（去掉走路的晃動），而且要「低於 −0.6 持續 0.4 秒」才翻、「高於 0.6」才翻回來，中間這段不動作，這叫**遲滯**。

轉動動畫　途中的角度用「先慢、後快、再慢」的曲線（ease-in-out），而且邊轉邊縮小，免得 240×135 的長方形轉到 90° 時四個角被切掉。縮放比例＝min(240 / (240|cosθ|＋135|sinθ|)，135 / (240|sinθ|＋135|cosθ|))。

失敗路徑：模擬器會轉，真機不會轉

嘗試：轉動時另開一張 240×135×16 位元（64.8 KB）的暫存畫面，把原畫面旋轉後畫進去再送螢幕。失敗：模擬器在瀏覽器裡跑，記憶體多得是，驗收全過；真機用 `perf` 量才發現「最大的一塊連續記憶體」只有 57 KB，配置失敗，程式就退回「不轉」。改用：停在 0° 或 180° 時直接用螢幕控制器的**硬體旋轉**（不佔記憶體），只有轉動那 0.35 秒用一張 8 位元色（32 KB）的暫存畫面。真機自測每格 23 ms，順暢。學到：模擬器測得到「邏輯」，測不到「資源限制」；凡是要配置大塊記憶體的功能，一定要在真機上量。

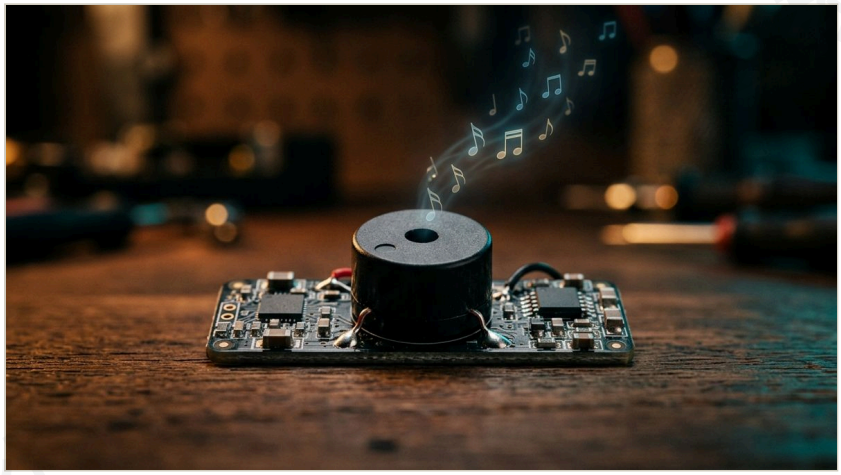
失敗路徑：修好記憶體之後，真機還是不能用 → 拿掉

嘗試：改用硬體旋轉後，AI 在桌上用序列埠自測（模擬按鍵、讀回角度）都正常。失敗：老師實際拿在手上仍然無法正常翻轉。真正的原因沒有查清楚，可能是感測器在外殼裡的方向與校正姿勢不合、平常握法的傾角落在遲滯區間裡、或門檻與持續時間不適合手持。改用：不再追查，整個功能拿掉；程式、設定項目、網頁 API 欄位與 6 項自動驗收一起刪除，只保留硬體旋轉的 HAL 函式。學到：「桌上自測通過」不等於「使用者拿在手上能用」，驗收要用真實的使用方式；一個常常失靈的功能比沒有這個功能更糟，修不好就該停損。



「最大連續記憶體」的比喻：空位加起來很多，卻沒有一整排連在一起，大巴士（64.8 KB 的暫存畫面）還是停不進去（AI 生成示意圖）

13.5　音樂



無源蜂鳴器：程式送多快的方波，它就發多高的音（AI 生成示意圖）



播放中的畫面：曲名、進度條、音量（關 / 小 / 中 / 大）。網頁儀表板也有「音樂」分頁可以點歌。

曲目	出處
小星星、兩隻老虎	法國民謠
生日快樂	Hill 姊妹，1893
歡樂頌、給愛麗絲	貝多芬
小蜜蜂	德國民謠
瑪莉有隻小綿羊、倫敦鐵橋	英美童謠
聖誕鈴聲	Pierpont，1857

按鍵怎麼用：同一組按鍵在不同狀態有不同意思。停止時，上下鍵選歌、同按開始播放；播放中，上鍵大聲、下鍵小聲（關 / 小 / 中 / 大四段，會記住），同按停止。播放中按鍵不發「嗶」聲，免得

打斷旋律。這種「依狀態改變按鍵意義」的設計叫**模式**（mode）：按鍵少的裝置都靠它，但畫面底部一定要提示現在每個鍵做什麼，使用者才不會迷路。

為什麼只有這幾首：都已經進入**公眾領域**（作者過世夠久，或本來就是不知作者的民謠），可以自由放進教材與公開網站。流行歌、動畫與電玩的主題曲還在著作權保護期內，即使只用蜂鳴器「嗶」出旋律，散布出去仍有侵權風險。

曲譜怎麼寫（學生可以自己加歌）：每個音寫成「音名＋八度：拍數」，用空白分開，例如歡樂頌開頭是 `E4:1 E4:1 F4:1 G4:1`；升記號寫 `D#5`；休止符寫 `R:0.5`。程式用**十二平均律**把音名換成頻率：A4 定為 440 Hz，每升一個半音乘以 2 的 12 次方根（約 1.0595），所以 C5 ＝ $440 \times 2^{3/12} \approx 523$ Hz。

為什麼整首升高一個八度：小型無源蜂鳴器靠一片小金屬膜振動，大約 2～4 kHz 最響，C4（262 Hz）這種低音幾乎聽不見。程式把每個音的頻率乘以 2（升一個八度），旋律不變、聽起來清楚得多。每個音只發聲 90%，留 10% 空白，連續兩個同音才聽得出是「兩下」。

> **失敗路徑：上下鍵接反了兩天才發現**
>
> 嘗試：程式把 GPIO0 叫「左鍵」、GPIO35 叫「右鍵」，在模擬器上排成左右兩顆。失敗：真機橫著拿時兩顆鍵在同一側上下排列，GPIO35 在上面，程式卻把下面那顆（GPIO0）當「上一項」；老師一直用相反的方向按，到 v1.8 才提出來。改用：GPIO35＝上鍵、GPIO0＝下鍵，所有畫面提示改成「上 / 下」，數值一律上鍵加、下鍵減；深度睡眠仍用 GPIO35 喚醒（GPIO0 開機時按著會進燒錄模式，不能當喚醒鍵）。學到：按鍵的名字要照使用者**實際看到的位置**取，不要照程式或模擬器的排法；也要主動問使用者「哪裡用起來怪怪的」，不要等他自己說。

13.6　移除番茄鐘、量角器與水平儀

老師使用兩天後表示這兩個功能平常用不到。移除後主選單從「計步、番茄鐘、水平儀……」變成「計步、音樂、水平儀……」，遊戲卡帶剩擲筊、骰子、步數目標。七日紀錄原本每天會記「完成幾顆番茄」，現在只記步數；但儲存在 NVS 裡的資料格式不能改（否則舊資料讀不回來），所以那個欄位保留在結構裡、永遠是 0。這是軟體維護的常見做法：**使用者看到的功能可以刪，存檔格式要相容**。

v1.9.4 拿掉水平儀：老師用了一陣子後指出，每個學生把感測器裝進外殼的方向都不一樣（有的平貼、有的側立），同一個水平儀畫面在每台裝置上的讀數都不同，後續教材很難統一呈現。計步只看三軸合成的加速度，不受安裝方向影響，所以只拿掉水平儀；傾斜角的計算留在程式裡給遊戲卡帶當延伸題。這和量角器、螢幕翻轉是同一類問題：**凡是依賴「哪邊朝下」的功能，都要先統一安裝方向，或先做校正**。

14　計步演算法第二版：用真機錄音校正

老師回報「實際走路時步數跳很快」。這一章是整個專案最重要的一次除錯：我（AI）先犯了一個錯，靠老師提供的真值才修正過來。

14.1　沒有真值就不能校正：走路測試畫面



① 先選裝置放在哪：褲子口袋、拿在手上、綁在手腕、包包裡。



② 倒數 3 秒、嗶一長聲開始錄，畫面即時顯示步數；走完按任一鍵停止。



③ 用上下鍵輸入「實際走了幾步」（上鍵加、下鍵減），和錄音一起存起來（最多保留 5 段）。

要知道演算法準不準，需要兩樣東西：**原始加速度錄音**（才能離線重算）和**真值**（實際走了幾步）。走路測試會把每秒 50 筆的 X / Y / Z 加速度存進檔案系統，每筆 6 bytes，120 秒約 36 KB；接回電腦後用序列埠 `dump 1` 倒出來，用 `tools/replay_walk.cpp`（和韌體同一份 C++）重播，就能在電腦上改演算法、立刻知道步數會變多少。

> **「一步」是什麼？**
>
> 手機、手環的計步器都是「左腳或右腳**每落地一次**算一步」。若你習慣「同一隻腳落地才算一步」，數到的會是一般步數的一半——這正是 14.3 節那個錯誤的來源之一。

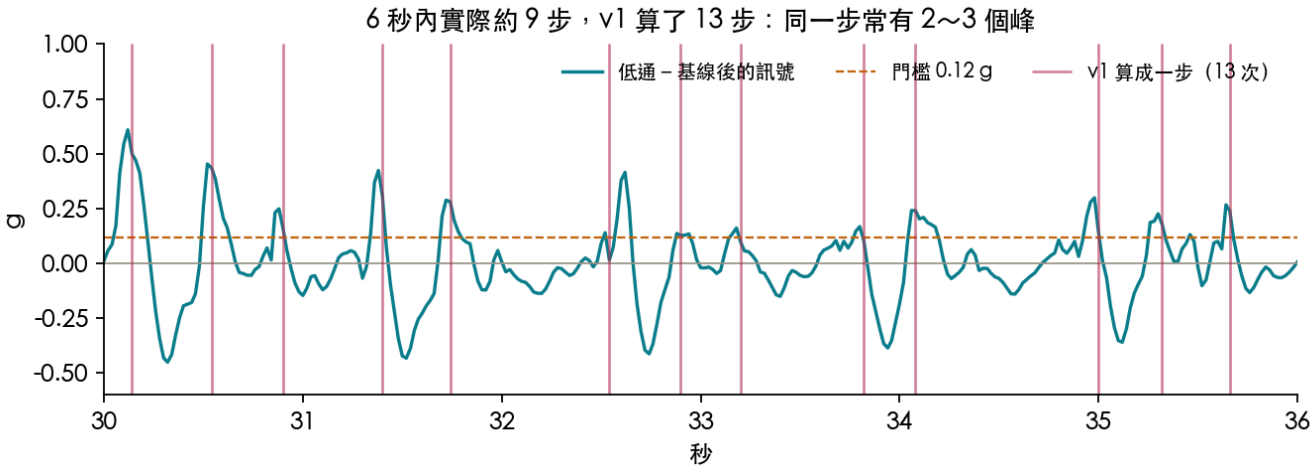


這次三段真機錄音都是把裝置放在褲子口袋走的（AI 生成示意圖）



手機、手環的「一步」：左右腳每落地一次算一步（AI 生成示意圖）

14.2　第一版為什麼多算



裝置放褲子口袋時，同一步常有 2～3 個波峰（腳跟著地、腿往前擺、另一腳著地的震動）。v1 只要求兩峰間隔 ≥ 250 ms，間隔 300～450 ms 的「第二個峰」就被算成另一步。

14.3　失敗路徑：我把諧波當成步頻

嘗試 → 失敗 → 改用 → 學到

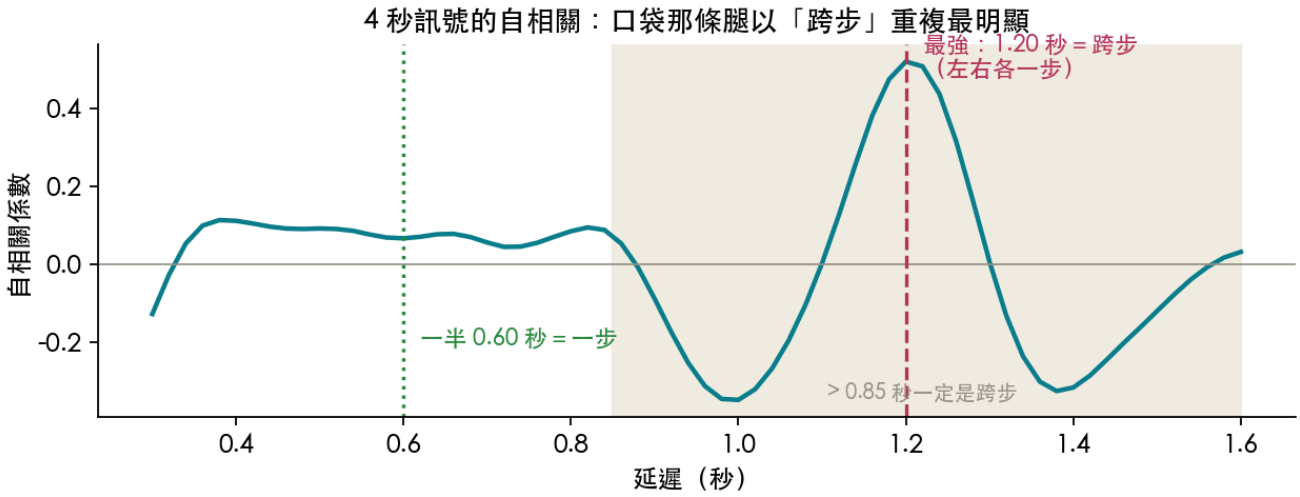
嘗試：第一段 90 秒錄音沒有真值，我用**頻譜**找最強的頻率，看到 1.74 Hz，就當成「每秒 1.74 步」，推算這段約 126～130 步，再把演算法調到接近這個數字（v2）。

失敗：老師用新的走路測試畫面走了「50 步」，裝置顯示 115。追問之後才知道老師是「同一隻腳落地才算一步」，也就是實際落地 100 次；而 64 秒走 100 步＝每步 0.64 秒，和 1.74 Hz 對不上。原來口袋那條腿的波形每一步有好幾個峰，1.74 Hz 是**諧波**（基本頻率的整數倍），不是步頻。我用錯的真值調出來的 v2 根本沒解決問題。

改用：請老師確認數法，用「100 次落地」這個確定的真值重新分析（14.4 節）。

學到：真值要問清楚「怎麼數的」，不能自己從頻譜推；頻譜上最高的那根，不一定是你要的那個頻率。

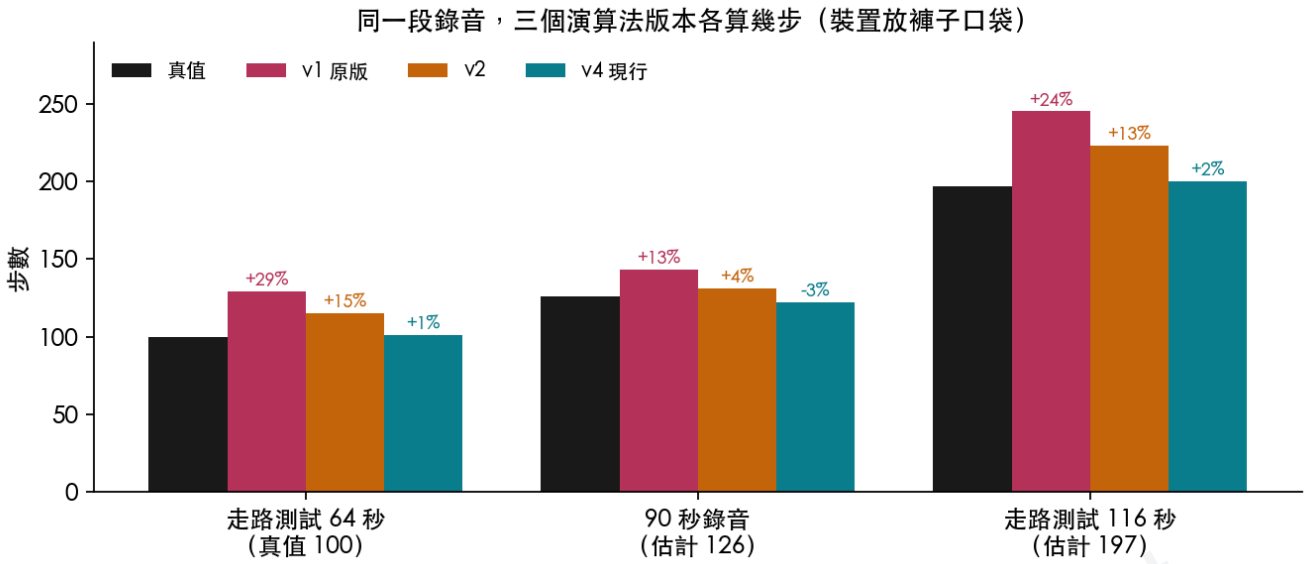
14.4　v4：重新上膛＋自相關步伐週期



把最近 4 秒的訊號和「自己往後挪一段時間的版本」相乘加總，就是**自相關**。挪到 1.20 秒時最像——那是口袋那條腿「跨一步」的時間（左右各一步），所以一步＝0.60 秒。

1. **重新上膛**：算進一步之後，訊號要先掉回基線（0 g）以下，才接受下一個峰。同一步的第二個峰通常還沒回到基線，就被擋掉。
2. **自相關估步伐週期**：每秒用最近 4 秒（200 筆）的訊號算延遲 0.30～1.60 秒的自相關，取最強的那個延遲。正常走路一步約 0.33～0.85 秒，所以最強延遲超過 0.85 秒，一定是「跨步」，步伐週期取一半。
3. **不反應期**：兩步之間至少要隔 max(300 ms，0.7 × 步伐週期)。走得慢時自動放寬，走得快時不會漏算。
4. **連續確認**（沿用 v1）：要連續 3 步才開始計數，一確認就一次補上 3 步——所以螢幕上的數字會「跳」一下，這是故意的，用來擋掉單次搖晃。

14.5　結果與回歸測試



三段真機口袋錄音、三個演算法版本。v4 在三段都在 ±3% 以內；v1 多算 13～29%。

三段錄音都放在 `tests/walk/`，指令 `make test-walk` 會用韌體同一份 C++ 重播，誤差超過 ±6% 就失敗。以後誰改了演算法，跑一次就知道有沒有退步——這叫**回歸測試**。目前只有「褲子口袋」的真機資料；拿在手上、綁在手腕、放包包的波形不一樣，還沒驗證，是留給學生的延伸題目。

趙珩宇　生活科技與語教材

15　效能與省電：先量再改

老師問「裝置還有多少容量？有發揮實際效能嗎？」。答案要用量的，不能用猜的。

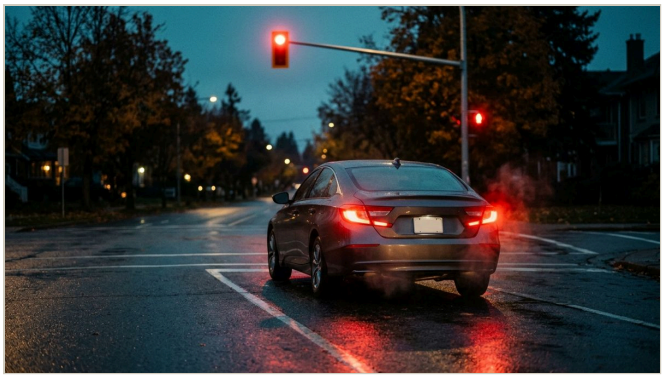
15.1　先做量測工具

工具（序列埠指令）	量什麼
`perf`	每秒主迴圈跑幾圈、CPU 活動比例、最長一圈幾毫秒、每秒送幾張畫面 / 略過幾張、FIFO 讀幾次、RAM、晶片溫度
`perf fs`	上面全部，加上韌體大小、檔案系統用量、設定區用量（這幾項要掃描，比較慢）
`plog`	每分鐘一筆寫進檔案（電壓、溫度、CPU、螢幕開關、Wi-Fi、睡眠時間），拔掉 USB 跑電池也記得到
`ps 0` / `ps 1`	同一份韌體把省電機制全關 / 全開，做前後對照（A / B 測試）
`key l / r / s / L`	由電腦模擬按上鍵、下鍵、同按、長按，讓測試可以自動跑（`tools/device.py`）
`sleeptest 秒`	接著 USB 時暫時「當作用電池」，才測得到 light sleep

第一次量（v1.5.1）：CPU 80 MHz、主迴圈每秒跑 **18,000 圈**，但真正在做事的時間每秒只有 207 ms，**其餘 79% 在空轉**——迴圈裡沒有任何「休息」，CPU 一直全速繞圈圈，這本身就在耗電、發熱。容量方面：韌體 1.33 MB / 程式區 1.97 MB、可用 RAM 186 KB、檔案系統用 36 / 128 KB，都還充裕。

失敗路徑：量測工具自己干擾了量測

嘗試：`perf` 順便回報韌體大小與檔案系統用量。失敗：每量一次，加速度計的資料就溢位一次。追查發現 `ESP.getSketchSize()` 每次都會把整個 1.3 MB 韌體從快閃記憶體讀一遍做驗證、查檔案系統用量要掃過全部檔案，各要數百毫秒，主迴圈被卡住。改用：一般 `perf` 只讀計數器，要看儲存資訊另外下 `perf fs`；而且下量測指令的那一圈不算進統計。學到：量測工具也是程式，也會佔時間；量之前要先確定工具本身夠「輕」。

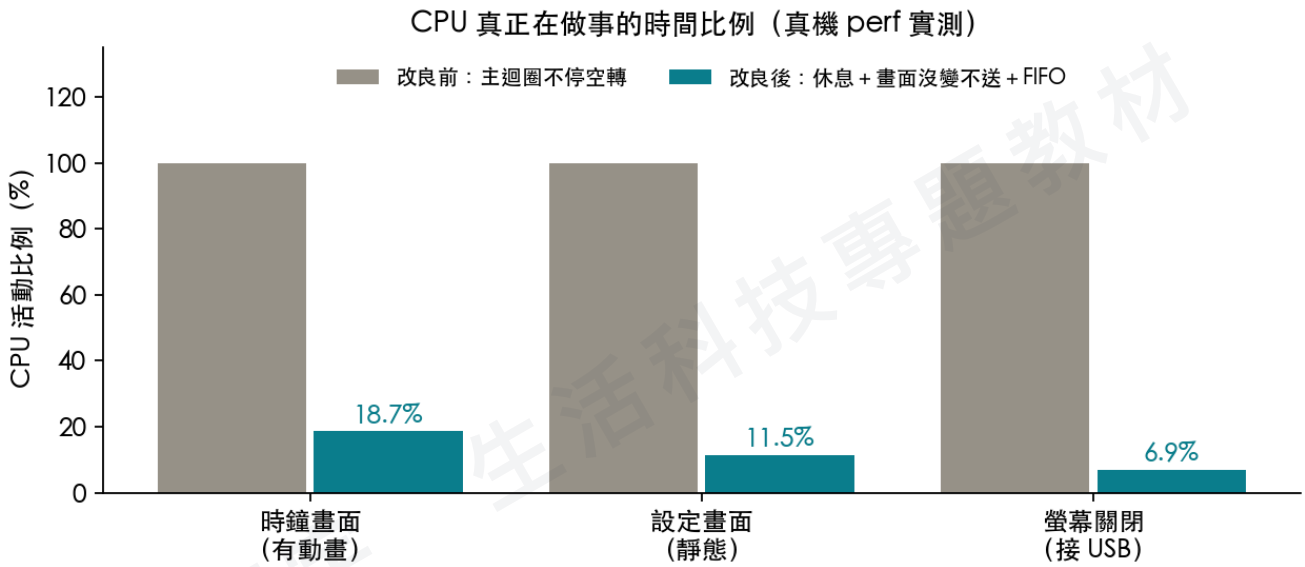


空轉的比喻：停紅燈時還一直踩油門，車沒動，油照燒（AI 生成示意圖）



FIFO 的比喻：信從上面放進去、從下面拿出來，先放的先拿；主程式每 0.1 秒來拿一疊（AI 生成示意圖）

15.2 五項改善與實測



真機實測。「改良前」主迴圈不停空轉，CPU 永遠 100% 在動。

項目	做法	實測
① 主迴圈休息	每圈做完事，就把剩下的時間交給作業系統（FreeRTOS），CPU 執行等待指令（WAITI）停下來等中斷	每秒迴圈 21,184 → 約 160 圈；時鐘畫面 CPU 100% → 18.7%
② 畫面沒變就不送	每張畫面畫好後算一個**雜湊值**，和上一張一樣就不送 SPI（送一張要 13 ms）	設定畫面每秒送畫面 8.8 → 0 張、CPU 11.5%
③ 加速度計 FIFO	ADXL345 自己存 32 筆資料，主程式每 0.1 秒一次讀回 10 筆，不必每 20 ms 叫醒一次	每批 10 筆、I2C 錯誤 0、沒有溢位
④ 自動關螢幕＋light sleep	沒按鍵 N 分鐘關背光；螢幕關、用電池、沒開無線時，讀 FIFO 的空檔進 light sleep	30 秒中睡了 28.0 秒（283 次）；關螢幕 CPU 6.9%

趣玩學 生活科技專題教材

⑤ 每分鐘步數紀錄	每天一個檔、每分鐘 1 byte、保留 30 天；儀表板畫每小時長條	跨日、重開機資料都在

失敗路徑：比對畫面的方法比送畫面還慢

嘗試：用晶片內建的 CRC32 比對整張畫面。失敗：算一次要 12 ms，幾乎和送一張畫面（13 ms）一樣久，省下來的時間被吃掉一半。改用：每 4 個位元組「左旋 5 位再 XOR」的簡單雜湊，約 1 ms。學到：CRC 是為了「抓傳輸錯誤」設計的，比對 UI 畫面不需要那麼強，夠用的最簡單方法通常最好。

15.3 加速度計的 FIFO 與輕敲偵測

暫存器	設定	意思
BW_RATE 0x2C	0x0A	每秒輸出 100 筆（輕敲偵測要夠快；計步用的 50 Hz 由相鄰兩筆平均得到，順便抗混疊）
FIFO_CTL 0x38	0x9F	串流模式：一直存最近 32 筆（100 Hz 下＝0.32 秒）
THRESH_TAP 0x1D	40	輕敲門檻 40 × 62.5 mg ＝ 2.5 g
DUR 0x21	32	一下輕敲最長 32 × 625 µs ＝ 20 ms（腳跟著地的衝擊比較長，會被擋掉）
LATENT / WINDOW	64 / 240	兩下之間要隔 80 ms 以上、300 ms 以內
INT_SOURCE 0x30	讀取	bit 5＝偵測到輕敲兩下（讀了就清除）。模組的中斷腳沒接到 ESP32，所以用「讀暫存器」代替中斷

失敗路徑：FIFO 還是溢位

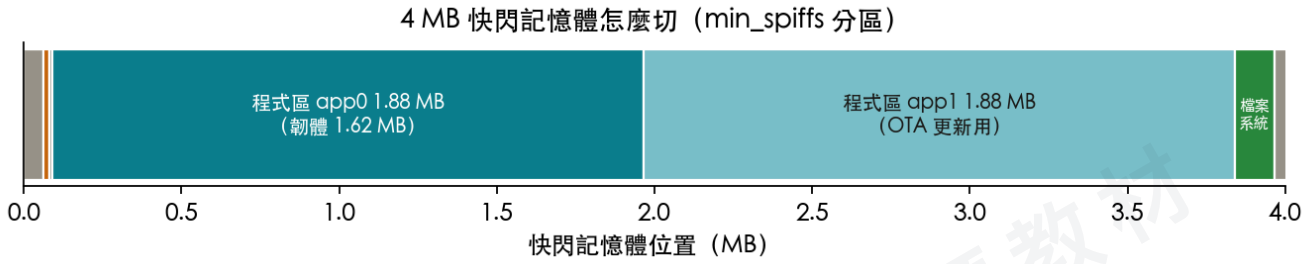
嘗試：每 0.2 秒讀一次 FIFO。失敗：FIFO 只存 0.32 秒，扣掉 0.2 秒只剩 0.12 秒餘裕；每分鐘寫一次檔案時主迴圈會卡超過 0.12 秒，資料就被蓋掉。改用：每 0.1 秒讀一次，餘裕變 0.22 秒。學到：緩衝區的大小要和「最慢的那件事」一起算，不能只看平均。

v1.13 起也支援 MPU6050，而且開機自動偵測：先找 ADXL345（I2C 位址 0x53），找不到再找 MPU6050（0x68），不必改程式或編譯參數。MPU6050 一樣用晶片內建的 FIFO（1024 bytes，50 Hz 可存 3.4 秒），只開加速度、陀螺儀設成待機省電，主程式每 0.1 秒讀一批。兩種感測器的單位不同（ADXL345 每 g 256、MPU6050 每 g 16384），驅動程式都換算成 g 再交給計步演算法，所以計步、拿起來亮螢幕的程式完全共用。

15.4　light sleep 的限制

ESP32 的 **light sleep** 會暫停 CPU 和大部分時脈，耗電降到約 1 mA，計時器或按鍵可以叫醒，記憶體內容都保留。但這次用的 Arduino 核心在編譯時**沒有打開自動電源管理**（設定檔裡沒有 `CONFIG_PM_ENABLE`），所以不能讓系統自己找空檔睡，只能在程式裡手動呼叫。另一個限制是背光的 PWM 訊號在睡眠時會停住，螢幕會閃爍，所以**只在螢幕關閉時才睡**；接著 USB 時也不睡，否則序列埠收不到指令。

15.5　快閃記憶體怎麼分配



4 MB 快閃記憶體：兩個 1.88 MB 程式區（一個在用、一個給 OTA 更新寫入）、20 KB 設定區（NVS）、128 KB 檔案系統（LittleFS）。

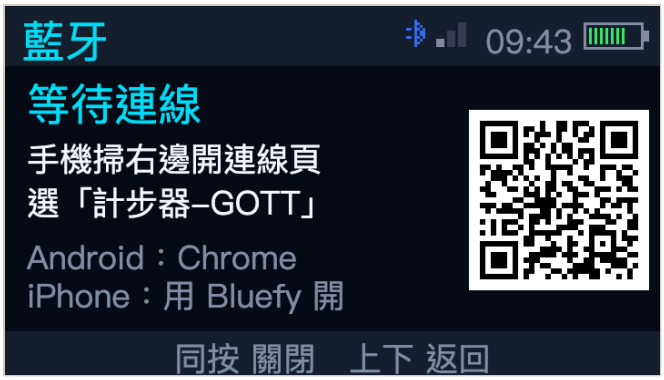
檔案系統裡放什麼	大小上限
每分鐘步數，30 天（每分鐘 1 byte，上限 255 步 / 分）	約 43 KB
plog 電源紀錄（1500 筆，約一天多）	約 30 KB
走路測試錄音（最多 5 段，空間不夠自動刪最舊的）	其餘

趙珩宇 生活科技專題教材

16　藍牙與網頁工具：手機、電腦怎麼連

Wi-Fi 熱點最耗電，藍牙（BLE）是比較省電的替代方案。另外，學生改了程式，現在可以完全在瀏覽器裡完成「模擬 → 燒錄」。

16.1　藍牙低功耗（BLE）與 GATT



設定 → 藍牙：開著時右邊是連線頁的 QR 碼，左邊顯示等待連線 / 已連線。



連線頁 ble.html：按「連線」選「計步器-GOTT」，即時看步數、電量、每小時步數，也能對時。

BLE 裝置把資料包成**服務**（service），每個服務裡有幾個**特性**（characteristic），每個特性有自己的編號（UUID）和能做的事：讀、寫、通知。手機連上之後就照編號讀寫，這套規則叫 **GATT**。

服務 / 特性	UUID	能做什麼
電池服務 / 電量	0x180F / 0x2A19（標準）	讀、通知：0～100%。這是藍牙官方定義的編號，手機系統看得懂
計步服務 / 步數	6e3a0002-…	讀、通知：uint32。步數一變，裝置主動推送
計步服務 / 每小時	6e3a0003-…	讀：24 個 uint16，今天每小時的步數
計步服務 / 對時	6e3a0004-…	寫：int32 秒數＋int16 時區分鐘
計步服務 / 資訊	6e3a0005-…	讀：文字（版本、電量、步伐週期、時間是否同步）



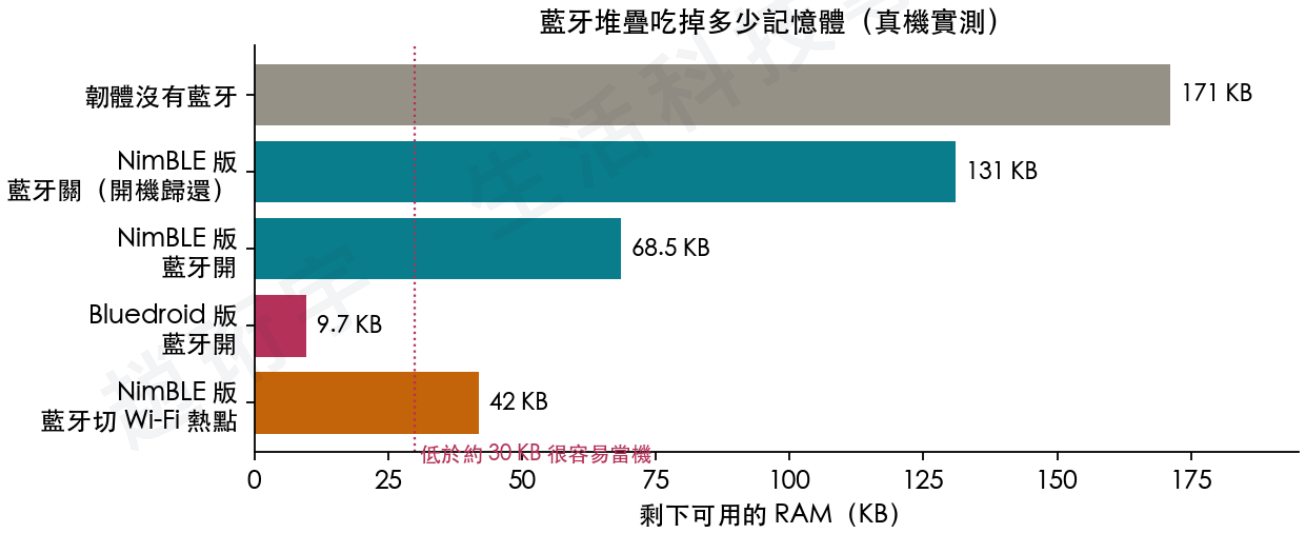
打開藍牙後，手機掃裝置畫面上的 QR 碼，連線頁就能看到即時步數（AI 生成示意圖）



Bluedroid 失敗的比喻：停車場幾乎滿了，藍牙這台大巴士停不進去（AI 生成示意圖）

手機怎麼連

- **Android**：用 Chrome 掃藍牙畫面上的 QR 碼，開啟 ble.html，按「連線」。手機的藍牙和定位要開（Android 規定掃描藍牙裝置需要定位權限）。可以「加到主畫面」當成 App 用。
- **iPhone**：Safari 和 iPhone 版 Chrome 都不支援 Web Bluetooth（Apple 的限制），要另裝免費的 Bluefy 瀏覽器開同一個網址；或改用 Wi-Fi 熱點。
- **電腦**：Chrome 或 Edge 開同一個網址。第一次 macOS 會詢問藍牙權限。



同一塊板子、不同藍牙堆疊，剩下的 RAM（真機實測）。

失敗路徑：藍牙一開，板子就一直重開機

嘗試：用 Arduino 內建的 Bluedroid 藍牙堆疊。第一個問題是程式太大（2.07 MB 塞不進 1.97 MB 程式區），老師選擇放棄 OTA、改用 3 MB 程式區。第二個問題才致命：開機時可用 RAM 只剩 104 KB，Bluedroid 初始化吃掉 94 KB，只剩 9.7 KB，下一次配置記憶體就失敗、當機；而且「藍牙開」已經存進設定，每次開機都重來一次。

改用：先燒一版不含藍牙的韌體救回來（步數、設定都還在），再改用較小的 **NimBLE** 堆疊：初始化只用 20 KB，開著還剩 68 KB；韌體也只有 1.60 MB，所以 OTA 又放得回去。另外加三個保險：藍牙關著時開機就把藍牙控制器保留的 41 KB 還給系統（之後要開藍牙會自動重開機一次）；從藍牙切到 Wi-Fi 時把藍牙整個卸載；開藍牙後 15 秒內當機，下次開機自動把藍牙關掉。

學到：可行性評估只量了「放不放得進快閃記憶體」，沒量「執行時要多少 RAM」——兩個都要量。

失敗路徑：一台連著，另一台就找不到

BLE 裝置被連上後預設就停止廣播，所以 AI 的測試程式連著時，老師的 Chrome 清單裡看不到裝置。改成連線後繼續廣播、最多同時 2 台。另一個小錯：插著 USB 時電量回報 0%，因為程式內部用 −2 代表「充電中」，被直接當成電量送出去；改成回報拔線前最後一次量到的電量。

趙珩宇 生活科技 試題教材

16.2　在瀏覽器裡寫程式、燒錄



「寫程式 / 燒錄」頁：左邊編輯器，右邊五步——模擬、編譯、連接、燒錄、序列埠監看。

1. **寫程式**：老師電腦執行 `make ide`，瀏覽器開 `http://localhost:8787/ide.html`。可編輯的檔案只有兩種（白名單）：學生卡帶 `cart_user.cpp`，和延伸專案的 `.ino`；其他檔案伺服器拒絕讀寫。
2. **模擬**：存檔後用 Emscripten 重編模擬器（約 20 秒），右邊的模擬器自動重新載入。
3. **編譯**：用 arduino-cli 編出真機韌體。瀏覽器本身不能編譯 C++，所以這一步一定在電腦上做。
4. **連接**：用 **Web Serial**（Chrome / Edge）選裝置。清單裡沒有裝置時，頁面依作業系統顯示驅動安裝說明：T-Display 的 USB 晶片是 CH9102，macOS 12 以後多半免驅動，Windows 要裝 WCH 官方的 CH343SER；也要確認是「資料線」不是純充電線。
5. **燒錄**：用 esptool-js 在瀏覽器裡把四個檔寫進板子（bootloader、分區表、boot_app0、程式），約 1 分鐘，設定與步數不會被清掉。



USB 線接上電腦，在瀏覽器裡就能編譯、燒錄（AI 生成示意圖）

T-Display S3：插電腦變隨身碟，匯出步數（v1.14）

S3 的 ESP32-S3 晶片內建 USB，可以假裝成一顆隨身碟（USB 大量儲存裝置，MSC）。在「七日紀錄」頁兩鍵同按 → 確認，電腦就會出現唯讀磁碟「PEDOMETER」，裡面有 `說明.txt`、`每日步數.csv`、`每小時步數.csv`、每天一個 `每分鐘_日期.csv`（最多 30 天），Excel 直接開，中文不亂碼（CSV 開頭有 UTF-8 BOM）。

做法	說明
內容「即時產生」	不先把檔案存一份：電腦要讀第幾個 512 bytes 的扇區，程式才算出那一段該是什麼（FAT12 檔案系統的開機區、檔案配置表、目錄、檔案內容）。每分鐘檔案每一行固定 22 個字元，第 n 分鐘就在第 18＋22n 個位元組，馬上算得出來。
平常不影響燒錄	平常 USB 是「序列埠＋燒錄」模式；只有按下匯出才切成隨身碟，結束時按任一鍵、存檔、切回、重新開機。
怎麼確認做對了	先在電腦上產生同樣的磁碟映像，交給 macOS 自己的 FAT 驅動掛載，五個測試檔（含 300 KB 與空檔）逐位元組相同、磁碟檢查沒有錯誤；再用實機匯出，三份 CSV 的步數互相對得上（每日 2,176＝每小時加總＝每分鐘加總）。

失敗路徑：結束後還是隨身碟

嘗試：按任一鍵就重新開機。失敗：① 隨身碟模式下螢幕照樣會自動關，按鍵只叫醒螢幕、畫面跳回時鐘，結束頁不見了；② 就算重開機，電腦仍認到隨身碟——ESP32-S3 切換 USB 的設定存在「重開機也不會清掉」的 RTC 區，只有按 RST 才回得來。改用：隨身碟模式下任何按鍵都是結束、畫面固定不關；重開機前先把 USB 切回序列埠模式並讓電腦重新辨識。學到：「重新開機」不等於「全部歸零」，要知道哪些硬體設定會留下來。

OTA：不用 USB 線也能更新

裝置開 Wi-Fi 熱點後，手機或電腦連上，開 `http://192.168.4.1/update` 上傳 `.bin` 檔（帳號 admin、密碼同設定頁）。OTA 是寫進「另一個」程式區，寫完、驗證通過才切過去開機；萬一寫到一半斷電，原本的程式還在。這就是為什麼分區表要有兩個一樣大的程式區。

趙珩宇　生活科技專題教材

附錄

A　腳位總表

訊號	T-Display	對接
I2C SDA	GPIO21	ADXL345 SDA
I2C SCL	GPIO22	ADXL345 SCL
3.3 V	3V3	ADXL345 VCC
GND	GND	ADXL345 GND、蜂鳴器 −
蜂鳴器	GPIO25	蜂鳴器 ＋
上鍵 / 下鍵	GPIO35 / GPIO0	板載（v1.8.1 起；之前程式把兩顆接反）
電池電壓	GPIO34（×2）	板載分壓，GPIO14 拉高才啟用
電池	JST 電池座	紅＋、黑− 直接插，不經開關

B　HTTP API（帳號 admin，密碼預設 1234，可在設定頁修改）

GET 只讀資料，不需要密碼；會改變裝置狀態的 POST 需要帳號密碼。手機連熱點後的網址是 `http://192.168.4.1`。

路徑	方法	內容
/	GET	中文儀表板（狀態、音樂、七日紀錄、設定）
/api/status	GET	steps、calories、distance_km、step_goal、music（正在播第幾首，−1＝沒播）、pitch、roll、is_level、battery_v、battery_pct、usb_power、wifi_on、time_synced、day_index、uptime_s、bmr_kcal、total_kcal_est、stride_cm、accel_ok、step_signal、step_period_ms、screen、screen_on、ble、walk_seq……
/api/live	GET	即時步數、訊號、步伐週期（給校正看即時變化）
/api/history	GET	七天的 day、steps（最後一筆是今天）
/api/minutes?day=N	GET	某一天每分鐘步數與每小時合計（預設今天，保留 30 天）
/api/walks	GET	走路測試清單：位置、裝置算的步數、輸入的真值、秒數、筆數

趙珩宇 生活科技專題教材

/api/music	GET / POST	GET：曲目清單與目前播放；POST：`action=play&i=N` 或 `action=stop`
/api/settings	GET / POST	stride_cm、weight_kg、height_cm、sex、age、step_goal、k_thresh、min_interval、max_interval、confirm_steps、brightness（30 / 50 / 100）、volume、theme、scr_off、psave、wifi_en、ble_en、web_pass、sta_ssid、sta_pass
/api/reset_steps	POST	今日步數歸零
/api/time	POST	epoch＝秒、tz_min＝時區分鐘（台灣 480）
/update	GET / POST	OTA 網頁更新：上傳 `.bin` 韌體（16.2 節）

C　指令

電腦端

```
make sim-build        # 重編模擬器（Emscripten）
node tools/qc_sim.mjs # 模擬器自動驗收（T-Display 75 項）
BOARD=s3 node tools/qc_sim.mjs   # T-Display S3 版（79 項，多心率等）
make flash-s3         # 編譯並燒錄 S3（燒錄前比對 MAC）
python3 tools/make_teacher_kit.py --check   # 產生給老師的燒錄包（附錄 J），並實際編譯兩個
Arduino 專案
make test-walk        # 三段真機口袋錄音的計步回歸測試（±6%）
make build            # 編真機韌體（先自動產生字型）
make flash PORT=/dev/cu.usbserial-XXXX   # 只接受允許清單中的埠
make monitor PORT=... # 看序列埠 115200
make ide              # 開瀏覽器寫程式 / 燒錄頁 http://localhost:8787/ide.html
python3 tools/sync_time.py --watch       # 裝置插上就自動回答 TIME?
```

裝置序列埠（115200 baud，打字後按 Enter）

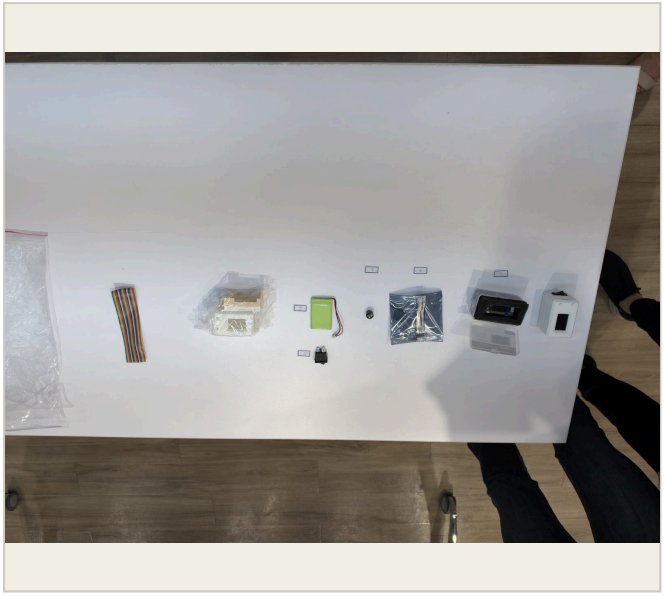
指令	做什麼
status / settings	印出目前狀態 / 所有設定（JSON）
get 名稱 / set 名稱=值	讀或改單一設定，例如 `set wifi_en=0`
time 秒數 時區分鐘	對時，例如 `time 1791357911 480`
perf / perf fs	效能量測（15.1 節）；fs 版含儲存空間，較慢
ps 0 / ps 1	省電機制全關 / 全開（A / B 測試）
plog / plog clear	印出 / 清除每分鐘電源紀錄

key l / r / s / L	模擬按左、右、同按、長按
ls / dump N	列出檔案 / 倒出第 N 段走路測試錄音
sleeptest 秒	接著 USB 時暫時當作用電池，測 light sleep
reboot / off	重開機 / 關機（深度睡眠，要按上鍵或重新燒錄才會醒）
i2c	掃描 I2C，列出找到的感測器位址（接線除錯）
inttest	（S3）測試 MPU6050、MAX30102 的中斷線有沒有接通
scroff	馬上關螢幕（測試拿起來亮螢幕）
tapstat / taplog / tapth 值	（S3）輕敲統計 / 每一下的大小與時間 / 調門檻（預設 0.85 g）

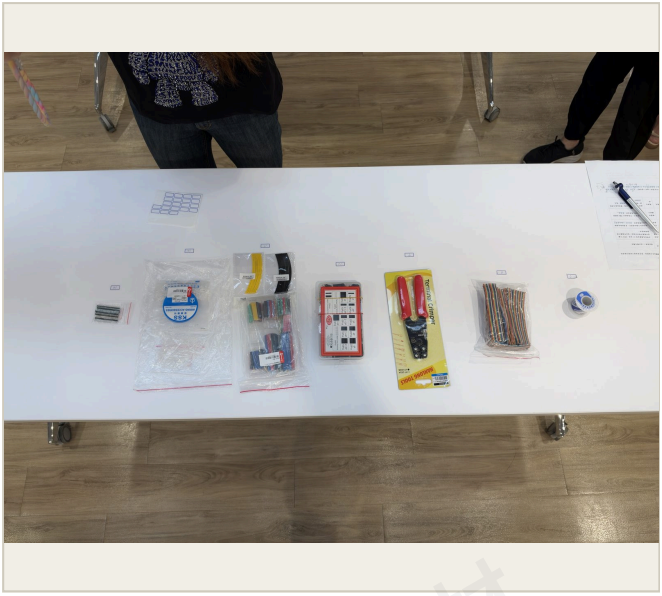
E　教學材料包（依原作 BOM 整理，一組一套）

編號	品項	用途	備註
1	LILYGO TTGO T-Display（ESP32）	主控＋螢幕＋充電	附排針，不必焊排針，直接焊線即可
2	GY-291 ADXL345 三軸加速度模組	計步、遊戲	只焊 VCC、GND、SDA、SCL
3	無源蜂鳴器（黑色圓殼）	提示音、旋律	先用 3.3 V 直流測：會響的是有源，退回換貨
4	603450 鋰電池 1000 mAh（JST 1.25 mm）	電源	避免尖物、不放電到 3.3 V 以下
5	KCD1 船型開關	（原作用）	**新版不使用**：關機改走設定選單的深度睡眠
6	焊錫（0.8 mm）	焊接	含松香芯
7	彩虹排線	感測器與蜂鳴器接線	剪四芯一段、兩芯一段
8、9	端子壓線鉗、杜邦母座與端子	想做可拆接頭時用	原作是直接焊，可省略
10	杜邦接頭套件	同上	
11	熱縮套管套件包	包覆排線焊點	防短路、防折斷
12	PC 塑膠圓頭十字螺絲 M3×10（每人 5 顆）	鎖外殼	
13	排針 1×20	備用	T-Display 已附

一	電烙鐵、烙鐵架、墊子	每組一套	手寫補充



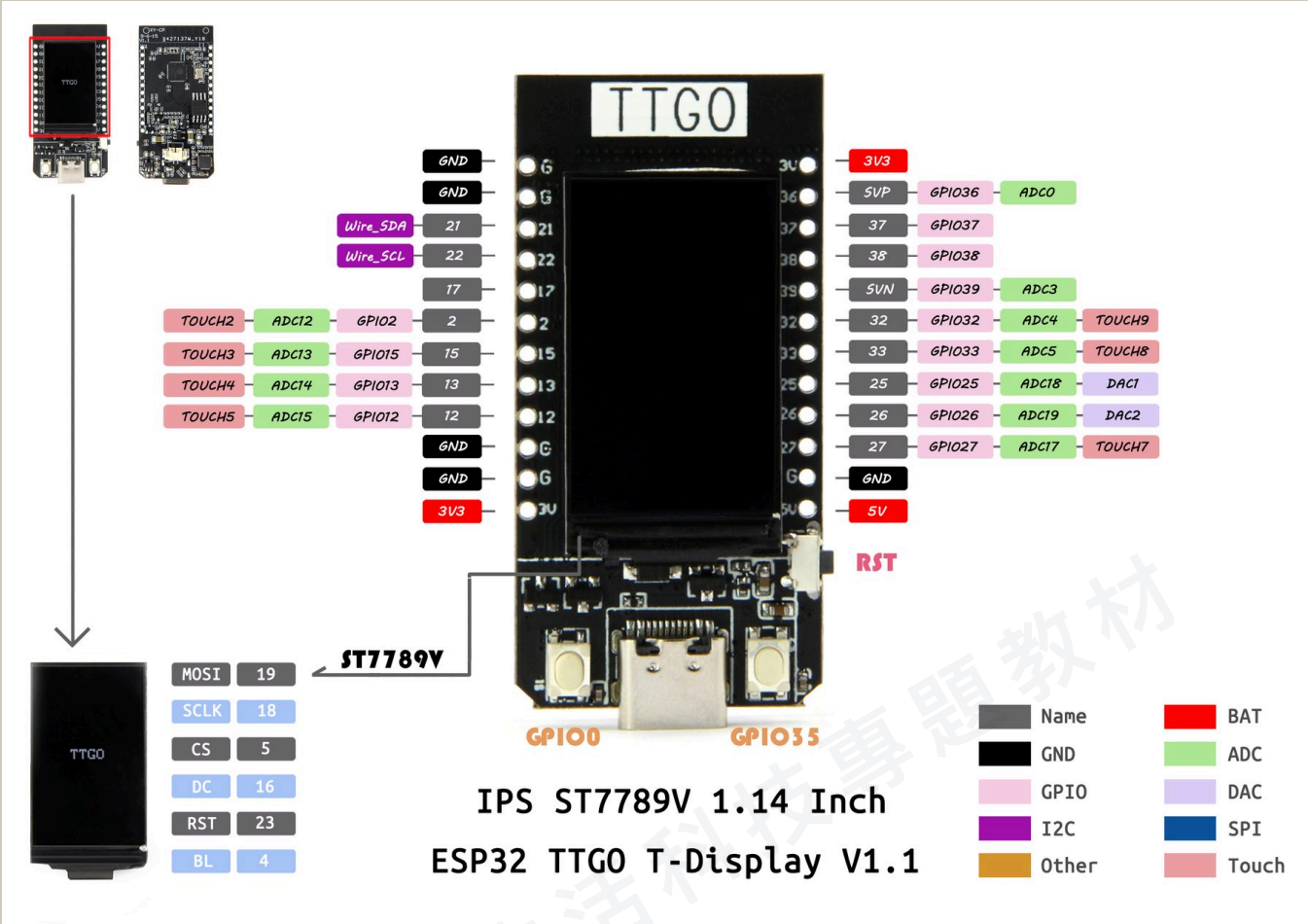
每組的主要零件（編號 1～5）＋排線與 3D 列印外殼。



共用工具與耗材（編號 6～13）：焊錫、排線、壓線鉗、杜邦套件、熱縮管、螺絲、排針。

趙珩宇 生活科技專題教材

F　官方資料與資料手冊



TTGO T-Display 官方腳位圖（LilyGO，MIT 授權，取自官方 GitHub）。電池量測用的 GPIO34（ADC_IN）與其開關 GPIO14（ADC_EN）是板內用途，腳位圖上 GPIO14 只標成 TOUCH4 / ADC14；ADC_EN 的用法要看官方範例程式與電路圖（docs/datasheet/）。

資料	來源	用途
T-Display 腳位圖、電路圖（ESP32-TFT(6-26).pdf）、TFT_eSPI 設定檔	github.com/Xinyuan-LilyGO/TTGO-T-Display（MIT）；電路圖副本在專案 `docs/datasheet/`	查腳位、背光與分壓電路、充電晶片
ESP32 Series Datasheet	espressif.com → Documentation → esp32_datasheet_en.pdf	GPIO 限制（34～39 只能輸入、無上拉）、開機模式腳、耗電模式
Neil Zhao, “Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer”, *Analog Dialogue* 44-06, 2010	analog.com → Analog Dialogue	計步演算法的經典參考：峰值偵測、時間窗、連續步伐判定、步幅估算

ADXL345 Data Sheet	analog.com → ADXL345 → Data Sheet	暫存器（0x2C BW_RATE、0x2D POWER_CTL、0x31 DATA_FORMAT、0x32 DATAX0）、3.9 mg/LSB、I2C 位址 0x53 / 0x1D
MPU-6000/6050 Register Map	invensense.tdk.com → MPU-6050 → Register Map	0x6B PWR_MGMT_1、0x1C ACCEL_CONFIG、0x3B ACCEL_XOUT_H、WHO_AM_I 0x75
ST7789V Datasheet	Sitronix（各代理商網站可下載）	135×240 的位移（offset 52 / 40）、Sleep In / Out 指令
TP4054 鋰電池充電 IC	各廠 datasheet（TopPower 等）	充電電流由板上電阻決定、充飽 4.2 V

資料手冊是「原廠說它會怎樣」，和說明書一樣要用實測對照：本講義 GPIO14 那一條就是查腳位圖才發現的。

G　硬體除錯流程

症狀	依序檢查
螢幕完全不亮	① 船型開關開了嗎？② 三用電表量電池：低於 3.3 V 先充電。③ 拔掉所有外接線、只插 USB：亮就是外接短路，不亮才是板子壞。
螢幕寫「找不到加速度計」	① GY-291 的 VCC / GND 接反或鬆脫？② SDA→21、SCL→22 有沒有對調？③ CS 腳是不是碰到 GND（會切成 SPI）？④ 燒一支 I2C 掃描程式看掃不掃得到 0x53。模擬器「接線」分頁可先演練 ①～③。
蜂鳴器沙沙聲或很小聲	① 用 3.3 V 直流短觸：連續嗶＝有源（拿錯了）；只「喀」一下＝無源（正確）。② 負極有沒有接到 GND？③ 是不是接到 GPIO25 以外的腳？
電量一直 0% 或亂跳	① 讀 GPIO34 前有沒有把 GPIO14 拉高？② 公式有沒有 ×2？③ 用電表量電池實際電壓對照。
燒錄失敗 Failed to connect	按住下鍵（GPIO0）→ 按一下 RST → 放開下鍵，再燒；USB-C 對 USB-C 線可能抓不到，換 USB-A 線。
模擬器正常、真機不正常	這就是模擬範圍之外的事：量電壓、看序列埠開機訊息、換線、換模組，一次只改一個變數。

趙珩宏 生活科技專題教材

H　審稿紀錄：另一版修訂稿的查證結果

2026-10-06 收到另一個 AI 產出的「修訂增補版」講義。逐條查證後，正確且有價值的已併入本版，錯誤的列於下表供對照（依本專案「修正版本說明格式」慣例）。

修訂稿的說法	查證	採用	修正後內容 / 理由
鋰電池安全守則（不同時剪兩線、熱縮管、3.0 V 下限）	正確	採用	第 2 節新增安全守則框
GPIO14 ADC_EN：100k:100k 分壓、約 21 μA 漏電、量完拉低	正確	採用	第 3.4 節；韌體同步改成量完拉低
活動消耗 vs 基礎代謝率的邊界	正確	採用	第 4.4 節框，補 Mifflin–St Jeor 例子
硬體除錯流程圖	正確	採用	附錄 G，改成表格以利列印
BMI270、ICM-42688-P 選型列	部分	修正後採用	ICM-42688-P 加速度雜訊是 **70 μg/√Hz**（datasheet），不是 0.28 μg/√Hz（差 250 倍）
ESP32 高電位 >2.64 V、低電位 <0.66 V	錯誤	修正	datasheet 是 0.75×VDD（≈2.5 V）與 0.25×VDD（≈0.8 V）
α=0.3 約等於截止 3～4 Hz；β=0.02 時間常數「數秒級」	錯誤	修正	$f_c = -\ln(1-\alpha)/(2\pi\Delta t)$：α=0.3 → 2.8 Hz；β=0.02 → 0.16 Hz、時間常數約 1 秒
演算法「參考 ADI AN-900」	編號有誤	修正	Neil Zhao 的文章是 *Analog Dialogue* 44-06（2010）「Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer」；ADI 的計步應用筆記是 AN-602 與 AN-2554。本演算法與該文同屬「峰值＋時間窗＋連續確認」一族，但非逐條實作
水平儀「增加中位數平滑濾波與動態水平校正」	與實作不符	不採用	本韌體水平儀用指數平滑（係數 0.2），沒有中位數濾波；歸零功能在「量角器」卡帶，不在水平儀
GY-291 上拉電阻 4.7 kΩ	未逐一查證	保留為「約」	常見模組為 4.7 kΩ，但各批次不同；講義只寫「板上有上拉電阻」
Pico 是「Cortex-M0+、Thumb-2」	部分	修正	Cortex-M0+ 用 ARMv6-M，主要是 Thumb 指令集（只含少數 Thumb-2 指令）；講義寫「ARM Thumb」
刪掉實拍照片、材料包、第 10 節遊戲化	—	不採用	修訂稿是純文字版，缺本版的照片與後續內容；以本版為主幹合併

J　給其他老師的燒錄方式

給其他老師的資料整理成「燒錄包」（`計步器燒錄包_v版本.zip`），裡面有完整的《老師操作說明》（5 頁）。三種方法燒出來的結果相同，平常更新韌體不會清掉步數與設定：

方法	用什麼	重點
一、網頁燒錄（建議）	電腦版 **Chrome / Edge**，免安裝	開 `henrychao521.github.io/pedometer-lab/flash.html` →「連接並燒錄」。**自動判斷** T-Display（ESP32）或 T-Display S3（ESP32-S3），不會燒錯板子
二、改程式後燒錄	**Arduino IDE 2.3 以上**＋esp32 by Espressif Systems 3.3.x＋程式庫 LovyanGFX、NimBLE-Arduino 2.5.1	燒錄包裡兩個專案資料夾依板子選一個。T-Display 要把 Partition Scheme 選 **Minimal SPIFFS**、Upload Speed 115200；S3 要把 USB CDC On Boot 選 **Enabled**
三、指令（大量）	Python 的 esptool	兩塊板子 bootloader 位址不同：ESP32 是 0x1000、ESP32-S3 是 0x0

驗證過的事

① 兩個 Arduino 專案用老師會選的同一套設定實際編譯成功。② 把 S3 寫回出廠韌體（模擬剛拆封的板子），再用網頁燒錄的同一套檔案燒，開機後檔案系統自動格式化、感測器自動偵測，可以直接使用。

D　圖片出處

- 封面、原作內部、T-Display 接線、ADXL345 特寫、零件、材料包、說明書：作者實拍（2026 年 10 月）。
- MEMS 原理示意、走路情境：AI 生成（FLUX.2 [klein] 4B，本機生成，Apache-2.0），為示意圖，非實物；中文說明為本文後製。
- LIS3DH 模組照片：SparkFun Electronics，CC BY 2.0，取自 Wikimedia Commons。
- GY-521 MPU-6050 模組照片：Nevit Dilmen，CC BY-SA 3.0，Wikimedia Commons。
- Raspberry Pi Pico：Phiarc，CC BY-SA 4.0，Wikimedia Commons。
- T-Display 腳位圖：Xinyuan-LilyGO，MIT License，github.com/Xinyuan-LilyGO/TTGO-T-Display。
- QR Code 編碼：Project Nayuki「QR Code generator library」（C 版），MIT License，nayuki.io；程式在 `firmware/app/third_party/`。
- 第 13～16 節的模擬器、網頁截圖與資料圖表（CPU 活動比例、演算法比較、自相關、RAM、分區）：本文以模擬器擷取、以真機量測數據繪製。
- 第 13～16 節的 12 張情境與比喻插圖（工作桌、手持裝置、熱像、口袋走路、腳印、紅燈、信件筒、停車場兩張、手機、筆電、蜂鳴器）：AI 生成（Google Gemini 圖像模型，經 Antigravity，2026-10-08），為示意圖，非實物或實

測。

- 音樂曲譜：作者依公眾領域旋律自行記譜。藍牙程式庫 NimBLE-Arduino：h2zero，Apache-2.0。
- 接線圖、架構圖、選型流程、逆向對照、計步訊號圖、模擬器與儀表板畫面：本文繪製 / 擷取。原作英文儀表板：學生作品，取自其韌體內嵌網頁，僅供教學比較。

本講義依著作權法第 46 條為學校授課需要重製他人著作之必要範圍內使用，各圖依授權條款標示來源。原作「GOTT Pedometer」為學生專題作品，本講義之新版韌體為另行設計，非原作之重製。

趙珩宇　生活科技專題教材

附錄 I　名詞解釋

依主題分組，同組內大致由淺到深。每個名詞先用一句白話說「它是什麼」，再說「為什麼這個專案需要它」，最後標出在哪一節出現。

I-1　硬體與電子

微控制器（MCU）	把 CPU、記憶體、計時器、各種輸入輸出腳做在同一顆晶片上的「小電腦」，專門控制一件事。本專案用的是 ESP32。和電腦的差別是沒有作業系統畫面、記憶體很小（KB 等級），但便宜、省電、開機就跑。〔第 2 節〕
ESP32	樂鑫（Espressif）出的微控制器，雙核心、最高 240 MHz、內建 Wi-Fi 與藍牙。本專案平常跑 80 MHz 省電，開熱點時升到 160 MHz。〔2、15 節〕
TTGO T-Display	LILYGO 公司把 ESP32、1.14 吋彩色螢幕、兩顆按鍵、USB 轉序列晶片與鋰電池充電電路做在一起的開發板。好處是不必另外接螢幕。〔2 節〕
GPIO	General Purpose Input/Output，通用輸入輸出腳。程式可以決定它是「輸出高 / 低電位」還是「讀取外面是高或低」。蜂鳴器接 GPIO25、按鍵在 GPIO0 / 35。〔3 節、附錄 A〕
ADC（類比數位轉換器）	把連續的電壓換成數字的電路。電池電壓經過板上的分壓電阻，由 GPIO34 的 ADC 讀成數字，再換算成電量。〔2、7 節〕
分壓電路	兩顆電阻串聯，中間點的電壓是總電壓的固定比例。4.2 V 的電池電壓超過 ADC 能量的範圍，所以先除以 2 再量。T-Display 要把 GPIO14 拉高才會接通這個電路。〔7 節〕
時間常數（RC）	電阻 × 電容，代表電容充到約 63% 要多久。分壓電路上有 1 µF 電容，時間常數約 50 ms，所以「拉高 GPIO14 後馬上讀」會讀到 0。〔7 節〕
I2C	只用兩條線（SDA 資料、SCL 時脈）就能和多顆晶片溝通的通訊方式，每顆晶片有自己的位址。ADXL345 的位址是 0x53。〔3 節〕
SPI	另一種通訊方式，用 4 條線、速度比 I2C 快很多，螢幕用它。送一整張 240×135 的畫面約 13 ms。〔15.2 節〕
上拉電阻	接在訊號線和電源之間的電阻，讓線在沒人拉低時保持高電位。I2C 一定要有，GY-291 模組板上已經有。〔3 節〕
加速度計 / ADXL345	量加速度的感測器。靜止時量到的是重力（1 g，朝下），走路時多了上下震動。ADXL345 是 ADI 公司的三軸數位加速度計，每格 3.9 mg。〔2、4 節〕
g（重力加速度單位）	1 g ≈ 9.8 m/s²。加速度計習慣用 g 當單位，mg 是千分之一 g。〔4 節〕
MEMS	微機電系統。加速度計裡面有一塊用彈簧懸吊的極小質量塊，加速時它會位移，晶片量電容變化換成加速度。〔5 節〕

暫存器	晶片內部一格一格的設定與資料欄位，每格有編號（例如 0x2C）。寫入暫存器＝設定晶片怎麼工作，讀暫存器＝拿資料或狀態。〔15.3 節〕
FIFO	First In First Out，先進先出的佇列。ADXL345 內建 32 格 FIFO，自己先存資料，主程式有空再一次讀回，像郵筒一樣，不必每 10 ms 跑去拿一次信。〔15.3 節〕
溢位（overflow）	緩衝區滿了還有新資料進來，舊資料就被蓋掉或丟掉。FIFO 只存 0.32 秒，主程式卡住超過這個時間就會溢位、漏掉步伐。〔15.3 節〕
中斷（interrupt）	硬體主動「叫」CPU 的訊號，CPU 會暫停手邊的事去處理。ADXL345 有中斷腳，但模組沒接到 ESP32，所以本專案改用「輪詢」：定時去讀狀態暫存器。〔15.3 節〕
輕敲偵測（tap detection）	ADXL345 內建的功能：加速度在很短時間內超過門檻就算一下「敲」，兩下間隔合適就算「輕敲兩下」。用來在螢幕關閉時叫醒裝置。〔13.3、15.3 節〕
無源 / 有源蜂鳴器	有源蜂鳴器內建振盪器，通電就「嗶」，只有一個音高；無源蜂鳴器沒有振盪器，要由程式送方波，方波多快、音就多高，所以能放旋律。本專案用無源的。〔2、13.5 節〕
方波與頻率（Hz）	方波是高低電位快速交替的訊號；每秒交替幾次就是頻率，單位 Hz。頻率越高聲音越高，A4＝440 Hz。〔13.5 節〕
PWM	脈波寬度調變：用很快的開關、改變「開著的時間比例」來控制平均亮度或功率。螢幕背光亮度 50% 就是一半時間開、一半時間關。〔13.3、15.4 節〕
背光	LCD 本身不發光，後面有一排 LED 照亮。背光是螢幕最耗電的部分，關掉背光、畫面就看不見。〔13.3 節〕
LCD 控制器 / 硬體旋轉	螢幕上有一顆晶片（ST7789）負責接收畫面並驅動像素。它可以設定「從哪個角落開始畫」，等於免費把畫面轉 90°、180°，不必在 ESP32 裡另外算。〔13.4 節〕
USB 轉序列晶片 / CH9102	ESP32 只會講序列埠（UART），電腦只認 USB，中間要一顆晶片翻譯。T-Display 用的是 WCH 公司的 CH9102，Windows 需要安裝驅動程式。〔16.2 節〕
驅動程式	讓作業系統認得某種硬體的軟體。沒裝 CH9102 驅動，電腦就看不到板子的序列埠，也就不能燒錄。〔16.2 節〕
DTR / RTS	序列埠的兩條控制線。T-Display 把它們接到 ESP32 的重置與開機模式腳，電腦打開序列埠時這兩條線抖一下，板子就會重開機。〔7 節〕
鋰電池、mAh	3.7 V 標稱、充滿 4.2 V、低於 3.3 V 應停用。mAh 是容量，1000 mAh 約等於以 100 mA 用 10 小時。〔2 節〕
mA（耗電流）	裝置每單位時間用掉的電量。開 Wi-Fi 約 100 mA 以上，light sleep 約 1 mA，差一百倍，這就是省電設計的意義。〔13.2、15.4 節〕

I-2　記憶體與儲存

RAM	執行時用的記憶體，斷電就消失。ESP32 有 520 KB SRAM（能放資料的約 320 KB），扣掉系統後程式能用約 130～180 KB，開 Wi-Fi 或藍牙還會再少。暫存畫面、藍牙、網頁伺服器都要從這裡借。〔13.4、16.1 節〕

堆積（heap）與最大連續區塊	程式執行中動態借用的 RAM 叫堆積。總共剩很多，不代表能借到一大塊：就像停車場有 10 個空位但不相鄰，大巴士還是停不進去。螢幕翻轉就是因為最大連續區塊只有 57 KB 而失敗。〔13.4 節〕
快閃記憶體（Flash）	斷電不會消失的儲存空間，放程式與資料。T-Display 有 4 MB。寫入次數有上限（約十萬次），所以不能每秒寫一次。〔15.5 節〕
分區表（partition table）	把 4 MB 快閃記憶體切成幾塊、每塊放什麼的清單：開機程式、設定區、兩個程式區、檔案系統。本專案用 min_spiffs 分區。〔15.5 節〕
USB 大量儲存裝置（MSC）	電腦看到就當成「隨身碟」的一種 USB 裝置類別，不用裝驅動。ESP32-S3 晶片內建 USB，可以假裝成隨身碟；原本的 T-Display（ESP32）只有 USB 轉序列埠晶片，做不到。〔16.2 節〕
FAT12 與扇區	最簡單的磁碟檔案系統格式，Windows、macOS 都看得懂。磁碟被切成一格格 512 bytes 的「扇區」，開頭是說明磁碟大小的開機區，接著是記錄「檔案放在哪幾格」的配置表（FAT）與目錄。本專案的隨身碟每一格都是電腦要讀時才算出來的。〔16.2 節〕
NVS	Non-Volatile Storage，ESP32 的「鍵—值」設定儲存區，像一本小筆記本：用名稱（例如 `steps`）存一個值。步數、設定、七日紀錄都放這裡。〔6、13.6 節〕
檔案系統 / LittleFS	在快閃記憶體上模擬「資料夾和檔案」的程式庫。LittleFS 是專為小型裝置設計、斷電不容易壞的版本。每分鐘步數、走路錄音放這裡。〔15.5 節〕
RTC 記憶體	ESP32 裡一小塊在深度睡眠時仍有供電的記憶體。關機前把時間寫進去，開機後接回，才不必每次重新對時。也用來記「上次是不是藍牙造成當機」。〔6.4、16.1 節〕
韌體（firmware）	燒進裝置、讓硬體動起來的程式。本專案的韌體約 1.62 MB。〔全書〕
位元組（byte）、KB、MB	1 byte＝8 位元，可以存 0～255。1 KB＝1024 bytes，1 MB＝1024 KB。每分鐘步數只用 1 byte，所以一分鐘最多記 255 步。〔15.5 節〕
8 位元色 / 16 位元色	每個像素用幾個位元表示顏色。16 位元有 65,536 色、一個像素 2 bytes；8 位元只有 256 色、1 byte，暫存畫面小一半，轉動動畫時看不太出差別。〔13.4 節〕
暫存畫面（sprite / frame buffer）	先在 RAM 裡畫好一整張畫面，再一次送到螢幕，避免看到畫到一半的閃爍。240×135×2 bytes ≈ 64.8 KB。〔13.4、15.2 節〕

I-3　軟體、程式架構與工具

主迴圈（loop）	Arduino 程式裡一直重複執行的那段程式：讀感測器、算步數、更新畫面、處理按鍵。改良前它每秒繞 18,000 圈卻大多沒事做。〔15.1 節〕
FreeRTOS	ESP32 上的即時作業系統，負責在多個工作之間切換。主迴圈「休息」就是把時間交還給 FreeRTOS，讓它在沒人要用 CPU 時讓 CPU 停下來等。〔15.2 節〕
CPU 活動比例	每秒裡 CPU 真正在執行程式的時間比例。100% 表示一刻不停；本專案改良後時鐘畫面約 19%。比例越低越省電、越不燙。〔15.2 節〕

light sleep / deep sleep	light sleep：暫停 CPU，記憶體保留，計時器或按鍵叫醒後從原地繼續，約 1 mA。deep sleep：幾乎全部斷電，只留 RTC，醒來等於重開機，晶片本身約 10 μA（整塊板子因穩壓器等會多一些）。「關機」用的是 deep sleep。〔15.4、1 節〕
HAL（硬體抽象層）	一張「硬體能做什麼」的函式清單（畫點、讀加速度、發聲……），功能程式只呼叫這張清單。換成真機或模擬器，只要換清單背後的實作。〔6.1 節〕
WebAssembly / Emscripten	WebAssembly 是瀏覽器能直接執行的機器碼格式；Emscripten 是把 C++ 編成 WebAssembly 的工具。所以同一份韌體程式能在網頁裡跑成模擬器。〔6、16.2 節〕
模擬器	在電腦上「假裝」成真實裝置的程式。能快轉時間、模擬走路與傾斜，但模擬不出記憶體不足、接觸不良這類硬體問題。〔6 節〕
序列埠（serial port）	一次送一個位元的通訊方式，USB 線接電腦後就是一個序列埠。程式印出的訊息、`perf` 等指令、對時都走這條路。速度單位是 baud（每秒位元數），本專案用 115200。〔附錄 C〕
arduino-cli	Arduino 的命令列版本，不必開視窗就能編譯、燒錄。網頁燒錄頁的「編譯」就是在電腦上呼叫它。〔16.2 節〕
燒錄（flash）	把編譯好的韌體寫進板子的快閃記憶體。寫入中途不能拔線，否則程式區會不完整、開機失敗（但設定區不受影響）。〔7、16.2 節〕
bootloader	開機時最先執行的小程式，負責看分區表、決定要從哪個程式區開機。燒錄時四個檔之一。〔16.2 節〕
OTA（Over-The-Air）	不接線、透過無線網路更新韌體。新韌體寫進「另一個」程式區，驗證通過才切換，所以失敗也不會變磚。〔16.2 節〕
雜湊（hash）／CRC	把一大串資料算成一個短數字，資料只要差一點，數字就幾乎一定不同。比對兩張畫面時只比雜湊就好。CRC 是專門抓傳輸錯誤的強力版本，但比較慢。〔15.2 節〕
XOR、位元旋轉	XOR（互斥或）是「兩個位元不同就得 1」的運算；位元旋轉是把一串位元往左推、推出去的從右邊補回來。兩者都極快，組合起來就是簡單雜湊。〔15.2 節〕
A / B 測試	其他條件都一樣，只改一個變因，比較兩組結果。`ps 0` / `ps 1` 就是在同一塊板子上把省電功能關或開來比較。〔15.1 節〕
回歸測試	每次修改後重跑一組固定的測試，確認以前對的地方沒有被改壞。`make test-walk` 用三段真機錄音做這件事。〔14.5 節〕
機器可判定的驗收	寫成「預期多少、實際多少」由程式自動判斷對錯的檢查，而不是人看畫面覺得「好像可以」。`qc_sim.mjs` 目前 68 項。〔6.3 節〕
白名單（允許清單）	只允許列出來的東西，其他一律拒絕。網頁編輯器只能改兩種檔、燒錄腳本只接受允許的序列埠，避免改到或燒到不該碰的東西。〔16.2 節〕
卡帶（cartridge）	本專案的遊戲擴充介面：每個小遊戲提供名稱、開始、更新、畫圖四個函式，放進清單就出現在「遊戲」選單。目前有擲筊、骰子、步數目標。〔10 節〕
遲滯（hysteresis）	「變過去」和「變回來」用不同門檻，中間留一段不動作的區域，避免在門檻附近來

	回跳。電量顯示用了它（已移除的螢幕翻轉也用過）。〔7、13.4 節〕
模式（mode）	同一個按鍵在不同狀態代表不同動作，例如音樂停止時上下鍵選歌、播放中上下鍵調音量。按鍵少的裝置都靠模式，但畫面要隨時提示每個鍵現在做什麼。〔13.5 節〕
轉盤選單（carousel）	把選項排成一圈，一次只看中間那一項，旁邊露出相鄰的項目，轉動時有動畫。比一次列出全部更適合小螢幕，也讓人知道「前後還有東西」。〔13.2 節〕
ease-in-out（緩入緩出）	動畫一開始慢、中間快、結束前又慢下來，看起來比等速自然。〔13.4 節〕

I-4　訊號處理與計步演算法

取樣率（Hz）	每秒量幾次。計步用 50 Hz（每 20 ms 一筆），足以描述每秒 1.5～3 步的走路；輕敲需要更快，所以晶片跑 100 Hz，再兩兩平均成 50 Hz。〔15.3 節〕
合成加速度	三軸加速度的長度 $\sqrt{(ax^2 + ay^2 + az^2)}$。不管裝置怎麼擺，長度都一樣，所以計步不必知道裝置方向。〔4.1 節〕
低通濾波	讓慢的變化通過、把快速抖動濾掉。本專案用最簡單的一階版本：每次只往新數值靠近 30%。〔4.1 節〕
基線（baseline）	訊號裡「慢慢變」的部分，主要是重力和姿勢。把它減掉，剩下的才是每一步的震動。〔4.1 節〕
截止頻率	濾波器開始把訊號壓小的頻率。α＝0.3、每 20 ms 時約 2.8 Hz，剛好保留走路的節奏。〔4.1 節〕
抗混疊（anti-aliasing）	取樣太慢時，快速的變化會被誤看成慢的（像影片裡車輪倒轉）。降低取樣率前先平均，可以避免這種假訊號。〔15.3 節〕
峰值偵測	訊號由上升轉成下降的那一點就是峰值。高過門檻的峰值才算「候選步」。〔4.1 節〕
門檻（threshold）	多大的峰值才算數。太低連桌子震動都算；太高輕輕走就算不到。預設 0.12 g。〔4.2 節〕
不反應期（refractory period）	算進一步之後，一段時間內不再接受新的峰值。借自神經科學：神經放電後也有一段時間不能再放電。v4 用 max(300 ms，0.7×步伐週期)。〔14.4 節〕
重新上膛（re-arm）	算進一步後，要等訊號先掉回 0 以下才接受下一個峰值，就像槍要上膛才能再發射。用來擋掉同一步的第二、第三個峰。〔14.4 節〕
連續確認	要連續出現幾個合格的步伐才開始算，一確認就把這幾步一次補上。擋掉單次搖晃，代價是起步時數字會延遲、然後一次跳 3。〔4.2、14.4 節〕
步頻、步伐週期	步頻是每秒（或每分鐘）幾步；步伐週期是一步花多少時間，兩者互為倒數。正常走路約每步 0.5～0.7 秒。〔14 節〕
頻譜（spectrum）	把訊號拆成不同頻率的成分，看每個頻率有多強。常用 FFT（快速傅立葉轉換）來算。〔14.3 節〕

諧波（harmonic）	基本頻率的整數倍。一步裡有好幾個峰的波形，頻譜上的 2 倍、3 倍頻也會很強，甚至比基本頻率還強，看最高的那根就會看錯。〔14.3 節〕
自相關（autocorrelation）	把訊號和「自己往後挪一段時間的版本」比相似度。挪到剛好一個週期時最像，所以可以找出「每隔多久重複一次」，不會被一個週期裡有幾個峰影響。〔14.4 節〕
延遲（lag）	自相關裡「往後挪多少時間」。本專案試 0.30～1.60 秒，找最像的那個。〔14.4 節〕
內積	兩個向量對應分量相乘再相加。兩個長度 1 的向量，內積＝夾角的 cos：同方向 1、垂直 0、反方向 −1。用來判斷裝置是正著還是倒著拿。〔13.4 節〕
真值（ground truth）	「實際上」的正確答案，例如實際走了幾步。沒有真值就無法知道演算法準不準，也不能拿來調參數。〔14.1 節〕
誤差百分比	(量到的 − 真值) ÷ 真值 × 100%。+13% 表示多算了 13%。〔14.5 節〕
步幅、基礎代謝（BMR）	步幅是一步的長度，用來把步數換成距離；基礎代謝是完全不動時一天消耗的熱量，用 Mifflin−St Jeor 公式估算。〔4.4 節〕

I-5　網路與無線通訊

Wi-Fi 熱點（AP 模式）	裝置自己當基地台，手機直接連它，不需要學校網路。代價是耗電與發熱。〔1、13.2 節〕
SSID	Wi-Fi 網路的名稱，例如「計步器-GOTT」。〔1 節〕
QR 碼（QR Code）	可以存文字的二維條碼。本專案用兩種：Wi-Fi 設定 QR（掃了直接連網）、網址 QR（掃了開藍牙連線頁）。〔1、16.1 節〕
強制門戶（captive portal）	手機連上 Wi-Fi 後自動跳出的網頁，飯店 Wi-Fi 登入頁就是這個機制。裝置把所有網址都導向自己的儀表板。〔1 節〕
HTTP、API	HTTP 是瀏覽器和伺服器溝通的規則；API 是程式之間約定好的「問法」，例如 `GET /api/status` 會回傳目前狀態。儀表板就是用 API 和裝置交換資料。〔附錄 B〕
JSON	一種用文字表示資料的格式，像 `{"steps":1234}`。人看得懂、程式也好處理，API 都用它回傳。〔附錄 B〕
GET / POST	HTTP 的兩種請求：GET 是「拿資料」，POST 是「送資料、要求改變」。會改資料的 POST 需要密碼。〔附錄 B〕
藍牙低功耗（BLE）	Bluetooth Low Energy，為了省電設計的藍牙版本，平常幾乎不耗電、需要時才短暫傳一點資料。比 Wi-Fi 熱點省電許多，適合手機持續看步數。〔16.1 節〕
藍牙堆疊（stack）／Bluedroid / NimBLE	處理藍牙通訊規則的整套程式。Bluedroid 功能完整但很大（吃 94 KB RAM）；NimBLE 只支援 BLE、小得多（約 20 KB），本專案改用它。〔16.1 節〕
廣播（advertising）	BLE 裝置定時對外喊「我在這裡、我叫什麼」，手機掃描時看到的清單就是收到的廣播。〔16.1 節〕
GATT、服務、特性	BLE 交換資料的規則。資料分成「服務」（一組相關功能），服務裡有「特性」（一個

趙玩一 二技專題教材

	個資料欄位），每個欄位可以讀、寫或通知。〔16.1 節〕
UUID	用來辨識服務與特性的編號。藍牙官方定義的用短編號（電池服務 0x180F），自訂的用 128 位元長編號，避免和別人撞號。〔16.1 節〕
通知（notify）	資料一改變，裝置主動推送給手機，手機不必一直問。步數特性就用通知。〔16.1 節〕
Web Bluetooth / Web Serial	讓網頁直接連藍牙裝置或序列埠的瀏覽器功能。Chrome、Edge 支援；Safari 與 iPhone 上的所有瀏覽器都不支援（Apple 限制），iPhone 要用 Bluefy。〔16 節〕
esptool-js	樂鑫官方燒錄工具的 JavaScript 版本，可以在瀏覽器裡透過 Web Serial 燒錄 ESP32。〔16.2 節〕
epoch 秒數、時區	從 1970 年 1 月 1 日 0 時（世界標準時間）起算經過幾秒，是電腦表示時間的通用方式。台灣時區是 +8 小時＝480 分鐘。對時就是送這兩個數字。〔13.1 節〕
ESP-NOW	樂鑫自家的點對點無線協定，不需要基地台，兩塊 ESP32 直接傳短訊息。延伸遊戲「擦身而過」會用到。〔10 節〕

I-6　音樂

音名與八度	C D E F G A B 對應 Do Re Mi Fa Sol La Si；後面的數字是第幾個八度，C4 是鋼琴中間的 Do。〔13.5 節〕
半音、升記號（#）	鋼琴上相鄰兩個鍵（含黑鍵）差一個半音，一個八度有 12 個半音。D#5 是比 D5 高一個半音的黑鍵。〔13.5 節〕
十二平均律	把一個八度平均分成 12 個半音，每個半音的頻率比都一樣（2 的 12 次方根）。現代鋼琴的調音方式。〔13.5 節〕
A4＝440 Hz	國際標準音高：鋼琴中央 Do 上面的 La 定為每秒振動 440 次，其他音都由它推算。〔13.5 節〕
BPM、拍	BPM 是每分鐘幾拍；1 拍的長度＝60000 ÷ BPM 毫秒。曲譜裡的 `:0.5` 就是半拍。〔13.5 節〕
休止符	不發聲的拍子，曲譜寫成 `R`。〔13.5 節〕
公眾領域	著作權保護期已過（台灣一般是作者死後 50 年），或本來就沒有著作權的作品，任何人都可以自由使用。〔13.5 節〕
著作權法第 46 條	學校為授課需要，可以在合理範圍內重製他人已公開發表的著作，但要註明出處。本講義的圖片出處依此標示。〔附錄 D〕

I-7　開發流程用語

唯讀備份	只讀不寫地把整片快閃記憶體複製下來，讀兩次比對一致才算數。任何會寫入的動作之前先做。〔8、11 節〕
反組譯	把機器碼翻回人看得懂的組合語言。ESP32 的程式和系統交織、工具支援弱，本專案

	評估後停下。〔8 節〕
SHA256	一種雜湊演算法，算出 64 個字元的「指紋」。兩次備份的 SHA256 一樣，代表內容一個位元都沒差。〔8、11 節〕
版本控制（Git / GitHub）	記錄每一次程式修改、可以隨時回到舊版的工具。原作的原始碼就是因為只存在一台電腦而遺失。〔8 節〕
失敗路徑	本講義的固定寫法：嘗試 → 失敗 → 改用 → 學到。失敗的原因往往比成功的做法更值得學。〔7、13～16 節〕
文件漂移	說明書寫的和實際程式做的慢慢不一致。拿到別人的專案時，先信實物與程式。〔1.1 節〕

趙珩宇　生活科技專題教材